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

RENESAS

Phase-out/Discontinued

IE-17K-ET

CLICE-ET
VERSION 1.6

Document No EEU-1466
(O. D. No. EEU-931)
Date Published January 1994 P
Printed in Japan

USER'S MANUAL

NEC

Phase-out/Discontinued

IE-17K-ET

CLICE-ET
VERSION 1.6

© NEC Corporation 1994

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.

PREFACE

Intended Readership :

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

Purpose :

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

SPC : 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 of IE-17K-ET 1-2
1.2.1 Interface with Target System 1-2
1.2.2 Program Memory 1-2
1.2.3 Emulation Method 1-2
1.2.4 Break Functions 1-2
1.2.5 Real-Time Trace Function 1-3
1.2.6 Data Memory Coverage Function 1-4
1.2.7 Program Memory Coverage Function 1-4
1.2.8 Other Features 1-5
1.3 Composition 1-6
1.3.1 System Diagram 1-6
1.3.2 Block Diagram 1-6

CHAPTER 2. SPECIFICATIONS 2-1

2.1 Console Interface 2-1
2.2 Environmental Conditions 2-1
2.3 Power Supply 2-1
2.4 Dimensions 2-1
2.5 Exterior Views 2-2
2.6 Accessories 2-4

CHAPTER 3. INSTALLATION 3-1

3.1 Switch Settings 3-1
3.2 Connector Connections 3-6
3.3 SE Board Installation 3-7
3.4 Connection to Host Machine 3-9
3.5 Connection to PROM Programmer 3-10
3.6 Connection to Target System 3-11

CHAPTER 4. STARTING 4-1

4.1 Communication with WINDOWS (Version 3.1) 4-2
4.1.1 Terminal Start-up 4-2
4.1.2 Settings 4-5
4.1.3 Program Download to IE-17K-ET 4-9

CHAPTER 5. DESCRIPTION OF COMMANDS 5-1

5.1 Prompt 5-3
5.2 Command Line Format 5-5
5.3 Command Buffer 5-7
5.4 Character Set 5-8
5.4.1 Special Control Characters 5-8
5.4.2 Special Characters 5-11
5.4.3 Dummy Character 5-13
5.5 Expression 5-15
5.6 Constant 5-17
5.7 Variables 5-18
5.7.1 Array 5-18
5.7.2 Q-Register 5-18
5.8 Built-In Macro Commands 5-20
5.8.1 Program Memory Control Commands............. 5-20
5.8.2 Data Memory Control Commands 5-42
5.8.3 Peripheral Circuit Control Commands 5-49
5.8.4 Emulation Commands 5-55
5.8.5 Break/Trace Condition Control Commands 5-64
5.8.6 Coverage Display Command 5-95
5.8.7 Help Command 5-98

CHAPTER 6. PROGRAM EXECUTION 6-1

6.1 Real-Time Emulation 6-1
6.2 Break Point Setting 6-2
6.2.1 Break by Program Address 6-4
6.2.2 Break by Data Memory Modification 6-8
6.2.3 Break by Multiple Break Condition 6-16
6.3 1-Step Emulation 6-18

Phase-out/Discontinued

CHAPTER 7. SE BOARD PROM CREATION 7-1

CHAPTER 8. ERROR MESSAGES 8-1

8.1 Command Errors 8-1
8.2 Hardware Errors 8-8

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-9
A.5 Description of Primitive Commands A-11
A.5.1 Pointer A-11
A.5.2 Function A-15
A.5.3 Assignment A-23
A.5.4 Argument Stack A-30
A.5.5 Q-Stack A-33
A.5.6 Macro A-36
A.5.7 Control A-42
A.5.8 Display A-49
A.5.9 Others A-56
A.6 Editor A-63
A.6.1 Command Buffer Editing A-63
A.6.2 Q-Register Editing A-63
A.6.3 Editor Commands A-64

APPENDIX B. BUILT-IN MACRO COMMANDS B-1

B.1 Program Memory Control Commands B-2
B.2 Data Memory Control Commands B-3
B.3 Peripheral Circuit Control Commands B-4
B.4 Emulation Commands B-5
B.5 Break/Trace Condition Control Commands B-6
B.6 Coverage Display Command B-6
B.7 Help Command B-7

List of Figures

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

List of Tables

Table No.	Title	Page
1-1	CLICE-ET and SIMPLEHOST Version Conjunction	1-1
5-1	Break/Trace Conditions Table	5-79
5-2	Trace State Transition	5-85
A-1	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-9

CHAPTER 1. GENERAL

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

1.1 GENERAL

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

The IE-17K-ET supports all the models in the 17K Series. A dedicated SE board is available for each model. The IE-17K-ET 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-ET can be operated alone by connecting it to a terminal. A more advanced debugging environment can also be realized by connecting the IE-17K-ET to the host machine and operating SIMPLEHOST^{TM}, which acts as the man-machine interface software between the IE-17K-ET and the operator.

Table 1-1 shows the version conjunction used for the IE-17K-ET interpreter CLICE (Command Language for In-Circuit Emulator) and SIMPLEHOST.

Table 1-1 CLICE-ET and SIMPLEHOST Version Conjunction

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

1.2 FEATURES OF IE-17K-ET

1.2.1 INTERFACE WITH TARGET SYSTEM

The actual product is used as the interface with the target system so that the electrical specifications as 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
- 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
- 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 OTHER FEATURES

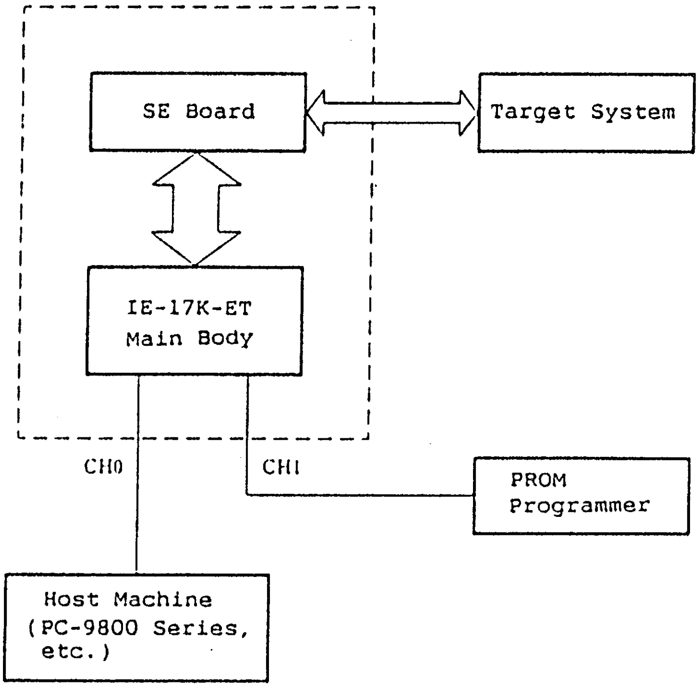
(1) The IE-17K-ET has two RS-232-C serial channels. Channel 0 is for console use and channel 1 is for PROM programmer, etc. use. A more advanced debugging environment can be realized by connecting channel 0 to a host machine PC-9800 Series and operating the man-machine interface software SIMPLEHOST.

(2) B5 size (25.7 cm x 18.2 cm x 6 cm) is extremely compact and easy to carry.

1.3 COMPOSITION

1.3.1 SYSTEM DIAGRAM

Figure 1-1 IE-17K-ET System Diagram

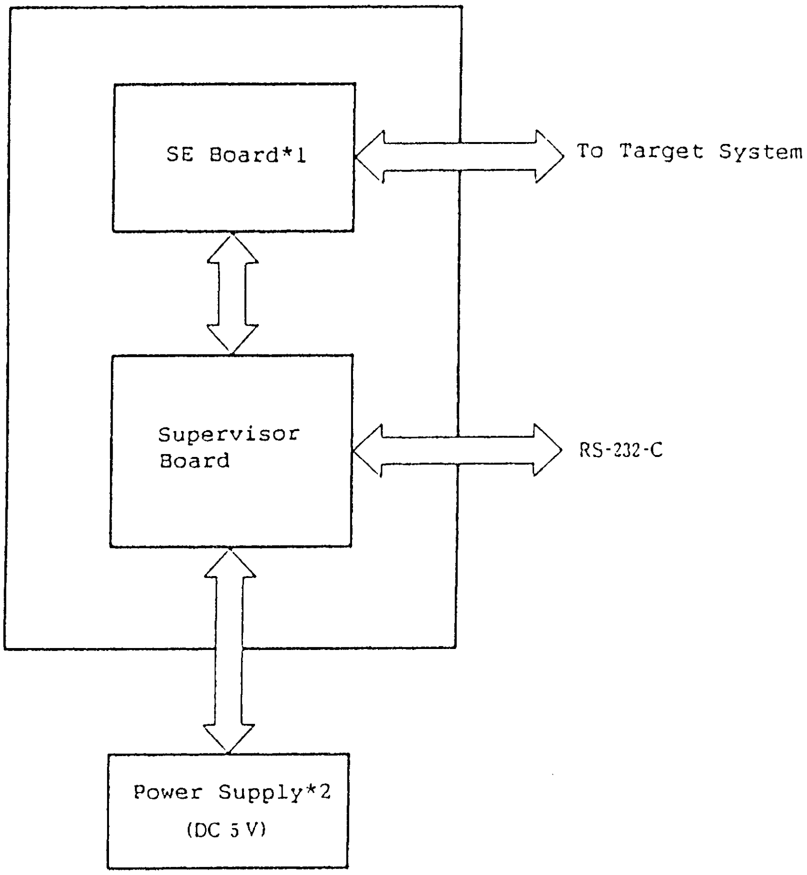

SE Board
Target System
IE-17K-ET
Main Body
CH0
CH1
PROM
Programmer
Host Machine
(PC-9800 Series,
etc.)

1.3.2 BLOCK DIAGRAM

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

- Cabinet (including connectors, switches, etc.)
- Supervisor (SV) board

Figure 1-2 IE-17K-ET Block Diagram


SE Board*1
To Target System
Supervisor Board
RS-232-C
Power Supply*2
(DC 5 V)

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

2: The IE-17K-ET does not incorporate a power supply, and therefore a separate commercially available DC power supply is necessary.

CHAPTER 2. SPECIFICATIONS

2.1 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

2.2 ENVIRONMENTAL CONDITIONS

Operating temperature range : 0 to +40°C
Storage temperature range : -10 to +50°C (no condensation)

2.3 POWER SUPPLY

DC 5 V ±5% 1.0 A (MAX.)
0.5 A (TYP.)

NOTE: The IE-17K-ET does not incorporate a power supply, and therefore a separate commercially available DC power supply is necessary.

2.4 DIMENSIONS

Cabinet dimensions 257 mm x 182 mm x 60 mm (excluding projecting parts)

2.5 EXTERIOR VIEWS

Figure 2-1 IE-17K-ET Exterior Views

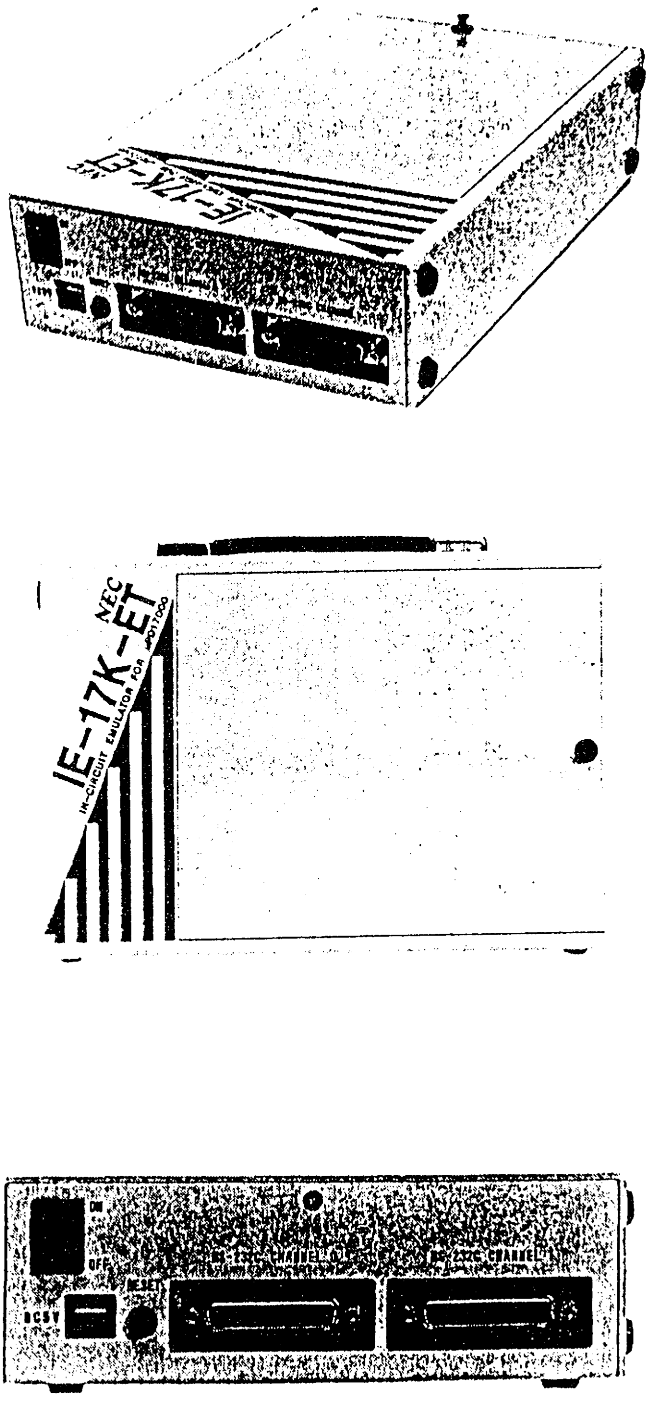
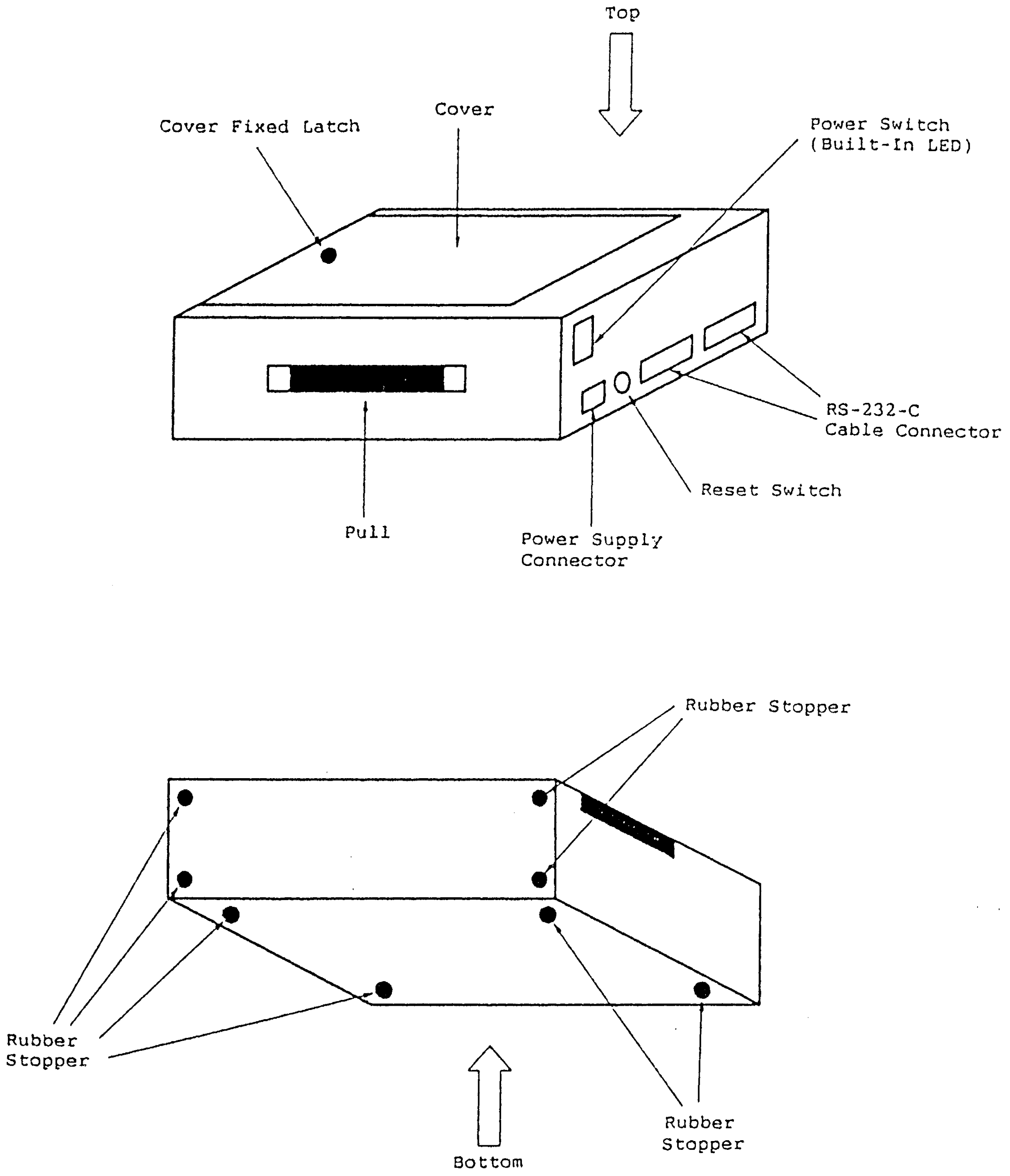

NEC
IE-17K-ET
IN-CIRCUIT EMULATOR FOR µPD17000
ON
OFF
DC5V
RESET
RS-232C CHANNEL 0
RS-232C CHANNEL 1

Figure 2-2 Nomenclature

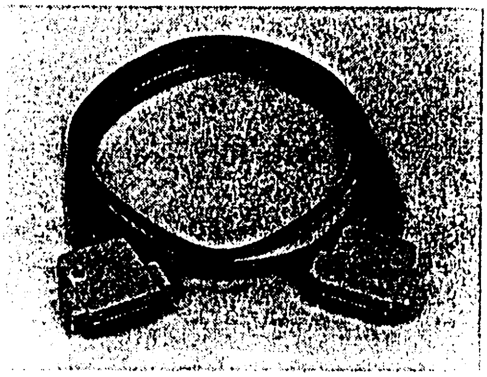

Top
Cover Fixed Latch
Cover
Power Switch
(Built-In LED)
RS-232-C
Cable Connector
Reset Switch
Pull
Power Supply
Connector
Rubber Stopper
Rubber
Stopper
Bottom
Rubber
Stopper

2.6 ACCESSORIES

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

(1) DC 5V power cable 1

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



(3) Others

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

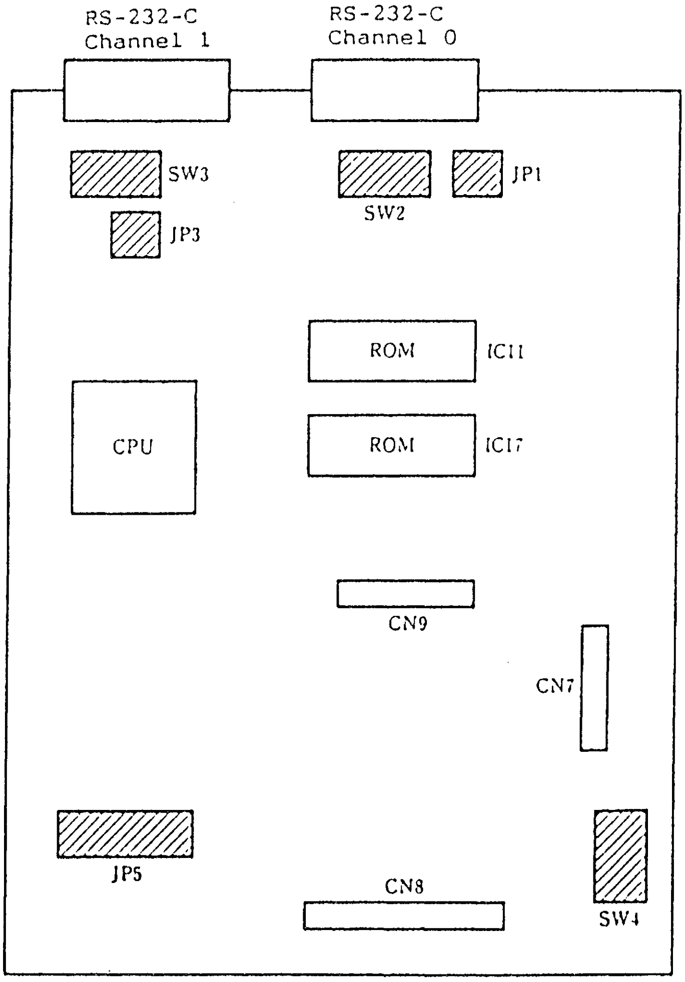
CHAPTER 3. INSTALLATION

This chapter covers settings of each switch on supervisor board connection with host machine and terminals required before IE-17K-ET start-up.

3.1 SWITCH SETTINGS

There is a supervisor board in the IE-17K-ET. Figure 3-1 shows configuration of switches on the supervisor board (The hatched area indicates switches).

Figure 3-1 Supervisor Board Switches Positions


RS-232-C
Channel 1
RS-232-C
Channel 0
SW3
JP3
SW2
JP1
ROM
IC11
ROM
IC17
CPU
CN9
CN7
JP5
CN8
SW4

NOTE: JP5 is already set. Do not change the setting.

Set the switches on the board as described below.

- RS-232-C control switches

  JP1, JP2, SW2 and SW3 are for RS-232-C control.

  JP1 and SW2 are for channel 0, and JP2 and SW3 are for channel 1.

  JP1 and JP2 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.

  JP1, JP2 .... Open
  SW2, SW3 .... Terminal mode

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

  SW4 is a DIP switch for RS-232-C setting.
  Figure 3-4 shows DIP switch settings.

Figure 3-2 RS-232-C Interface Circuit Diagram

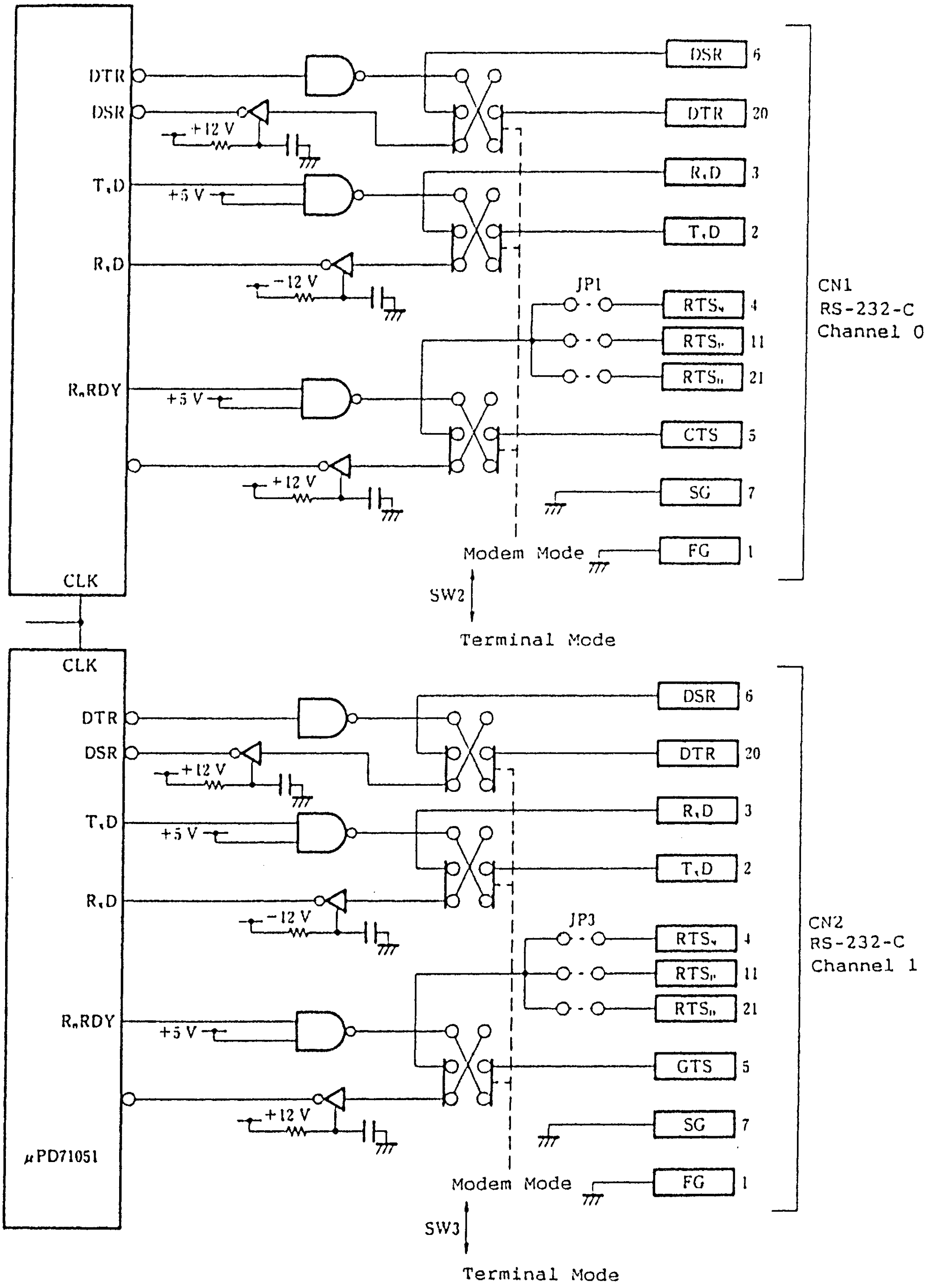

DTR
DSR
+12 V
TxD
+5 V
RxD
−12 V
RxRDY
+5 V
+12 V
CLK
DSR 6
DTR 20
RxD 3
TxD 2
JP1
RTS 4
RTS 11
RTS 21
CTS 5
SG 7
FG 1
CN1
RS-232-C
Channel 0
Modem Mode
SW2
Terminal Mode
CLK
DTR
DSR
+12 V
TxD
+5 V
RxD
−12 V
RxRDY
+5 V
+12 V
μPD71051
DSR 6
DTR 20
RxD 3
TxD 2
JP3
RTS 4
RTS 11
RTS 21
GTS 5
SG 7
FG 1
CN2
RS-232-C
Channel 1
Modem Mode
SW3
Terminal Mode

Figure 3-3 Accessory Cable Connection

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

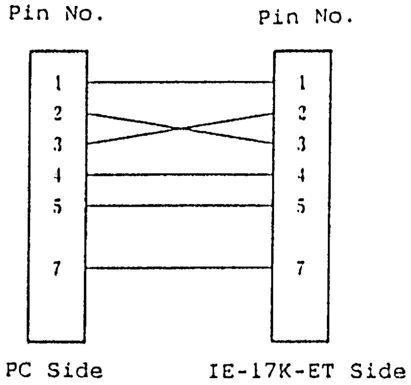
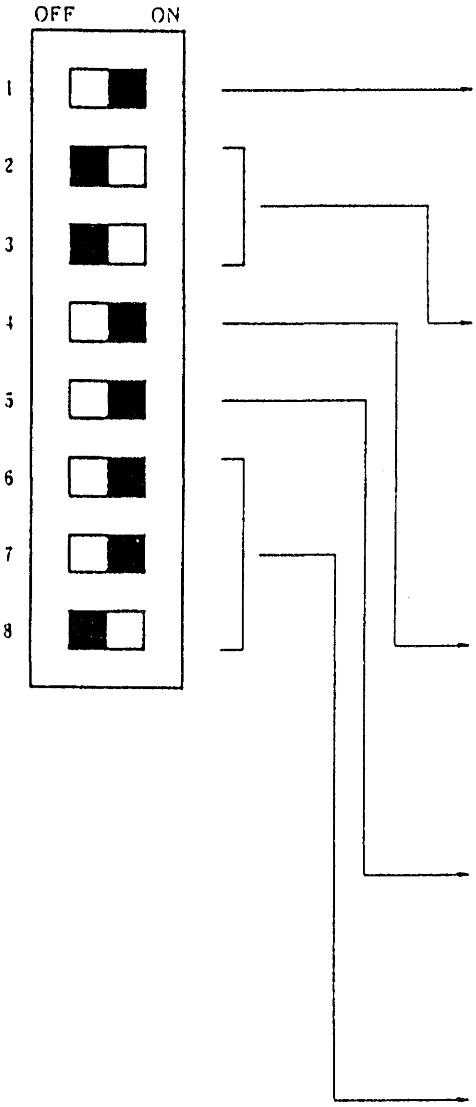

Pin No.
Pin No.
1
2
3
4
5
7
1
2
3
4
5
7
PC Side
IE-17K-ET Side

Figure 3-4 DIP Switch Settings



Control Method

1	Setting Contents
ON	Flow control
OFF	Line control

Parity Bit

2	3	Setting Contents
ON	ON	Odd
ON	OFF	Not allowed
OFF	ON	Even
OFF	OFF	None

Stop Bit

4	Setting Contents
ON	2 bits
OFF	1 bit

Character Length

5	Setting Contents
ON	8 bits
OFF	7 bits

Baud Rate

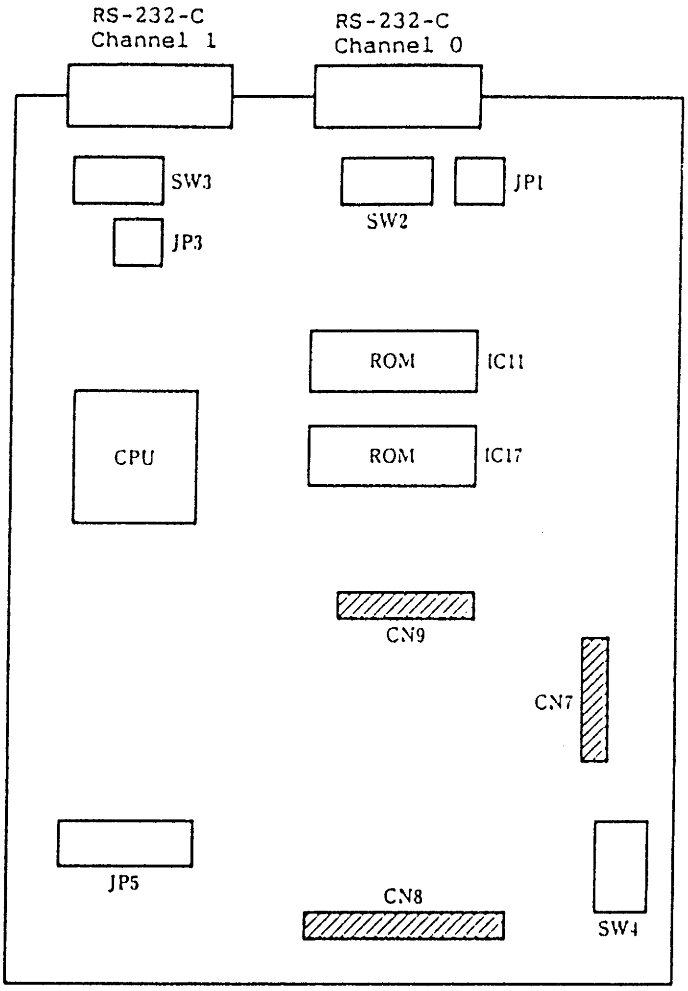
6	7	8	Setting Contents
OFF	OFF	ON	76800 bps
OFF	OFF	OFF	38400 bps
ON	ON	ON	19200 bps
ON	ON	OFF	9600 bps
ON	OFF	ON	4800 bps
ON	OFF	OFF	2400 bps
OFF	ON	ON	1200 bps
OFF	ON	OFF	600 bps

Remarks : The shaded area is factory setting

3.2 CONNECTOR CONNECTIONS

The layout of the three connectors CN7, CN8, and CN9 of the supervisor board connection is shown below (The shaded areas are connectors).

Figure 3-5 Connectors Layout (Supervisor Board)


RS-232-C
Channel 1
RS-232-C
Channel 0
SW3
SW2
JP1
JP3
ROM
IC11
CPU
ROM
IC17
CN9
CN7
JP5
CN8
SW4

Three connectors, CN7, CN8 and CN9 are used to mount the SE board.

3.3 SE BOARD INSTALLATION

When the IE-17K-ET is shipped, the supervisor board is installed to the IE-17K-ET as a control board. However, the SE board corresponding to the 17K Series model is not installed. Therefore, when developing the 17K Series, the SE board corresponding to the model must be installed to the IE-17K-ET separately from the IE-17K-ET.

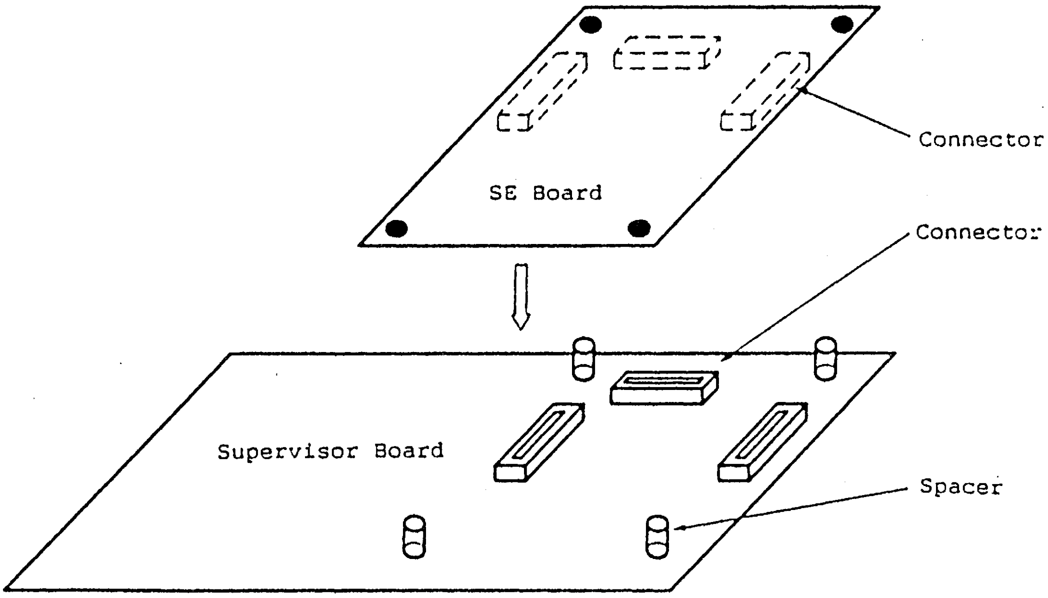
For a detailed description of the SE boards, refer to the user's manual for each SE board. Then installing an SE board to the IE-17K-ET, proceed as follows.

① Pull the cover latches at the top of the IE-17K-ET and remove the top outsider cover.

② Connect the connector on the supervisor 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 3-6 SE Board Installation


Connector
SE Board
Connector
Supervisor Board
Spacer

③ Fasten the SE board and the supervisor board with the screws.

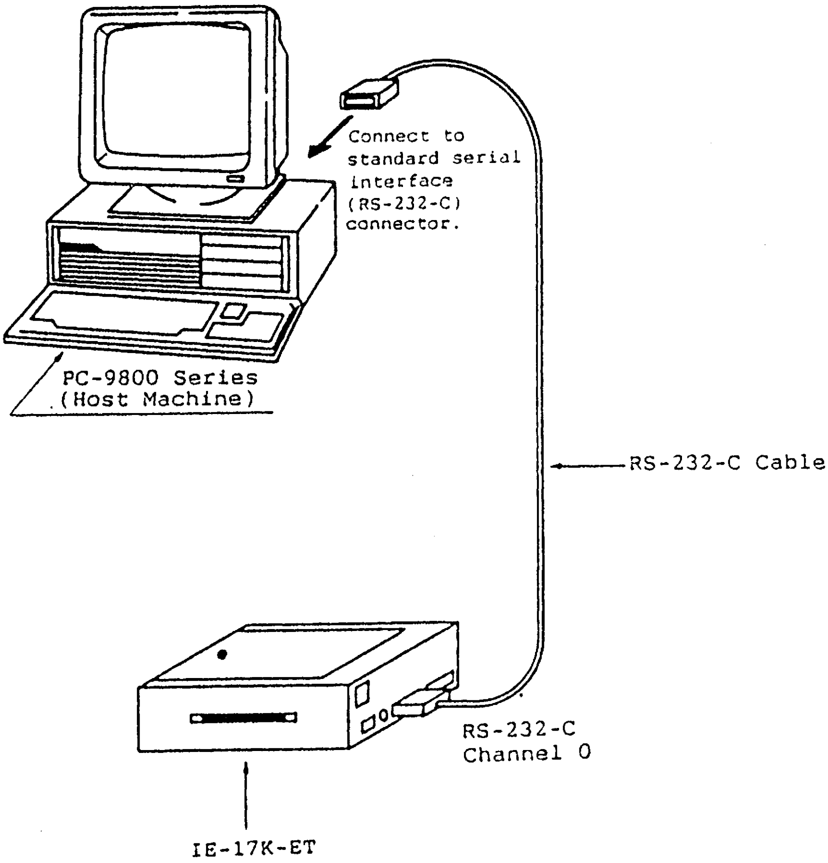
④ Reinstall the top inside and outside covers.

3.4 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-ET and PC-9800 Series power switch and connect the RS-232-C channel 0 connector of the IE-17K-ET 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-ET.

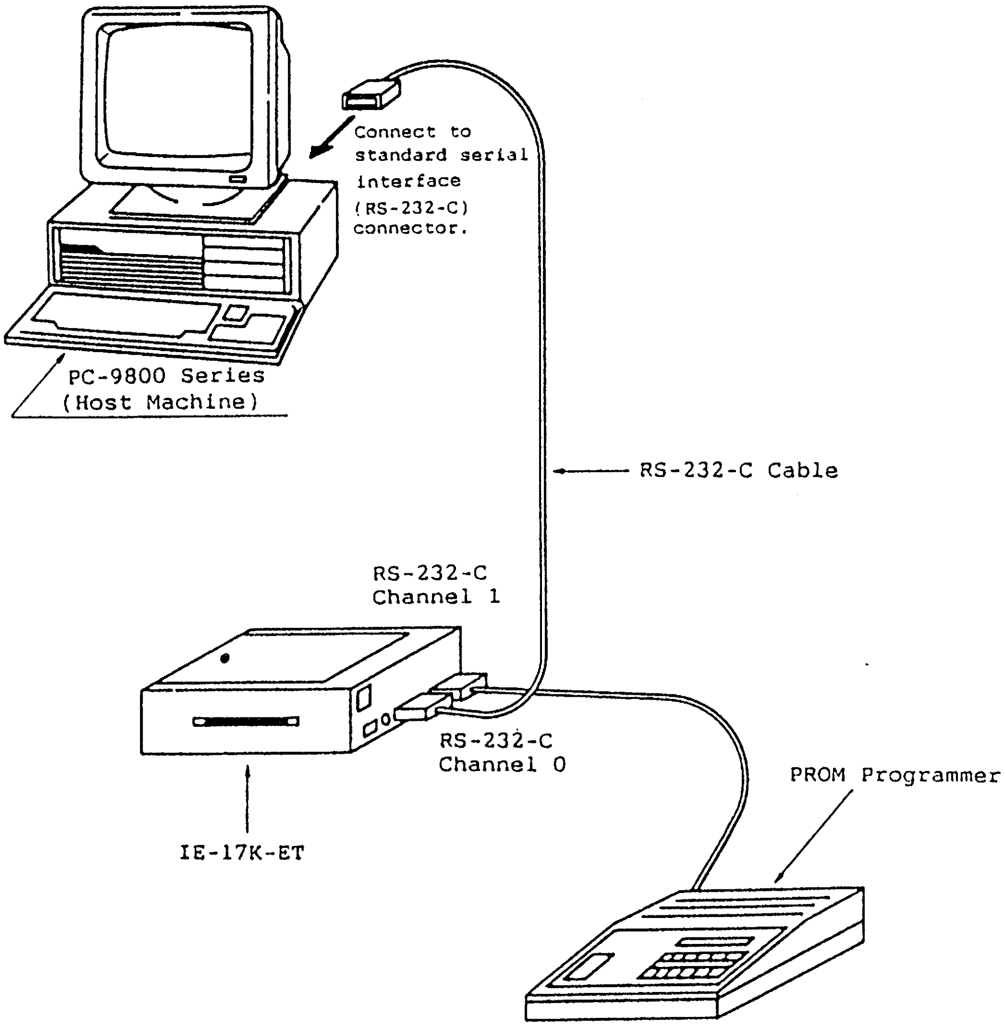
Figure 3-7 IE-17K-ET and PC-9800 Series Connection


Connect to standard serial interface (RS-232-C) connector.
PC-9800 Series (Host Machine)
RS-232-C Cable
RS-232-C Channel 0
IE-17K-ET

3.5 CONNECTION TO PROM PROGRAMMER

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

Figure 3-8 IE-17K-ET and PROM Programmer Connection

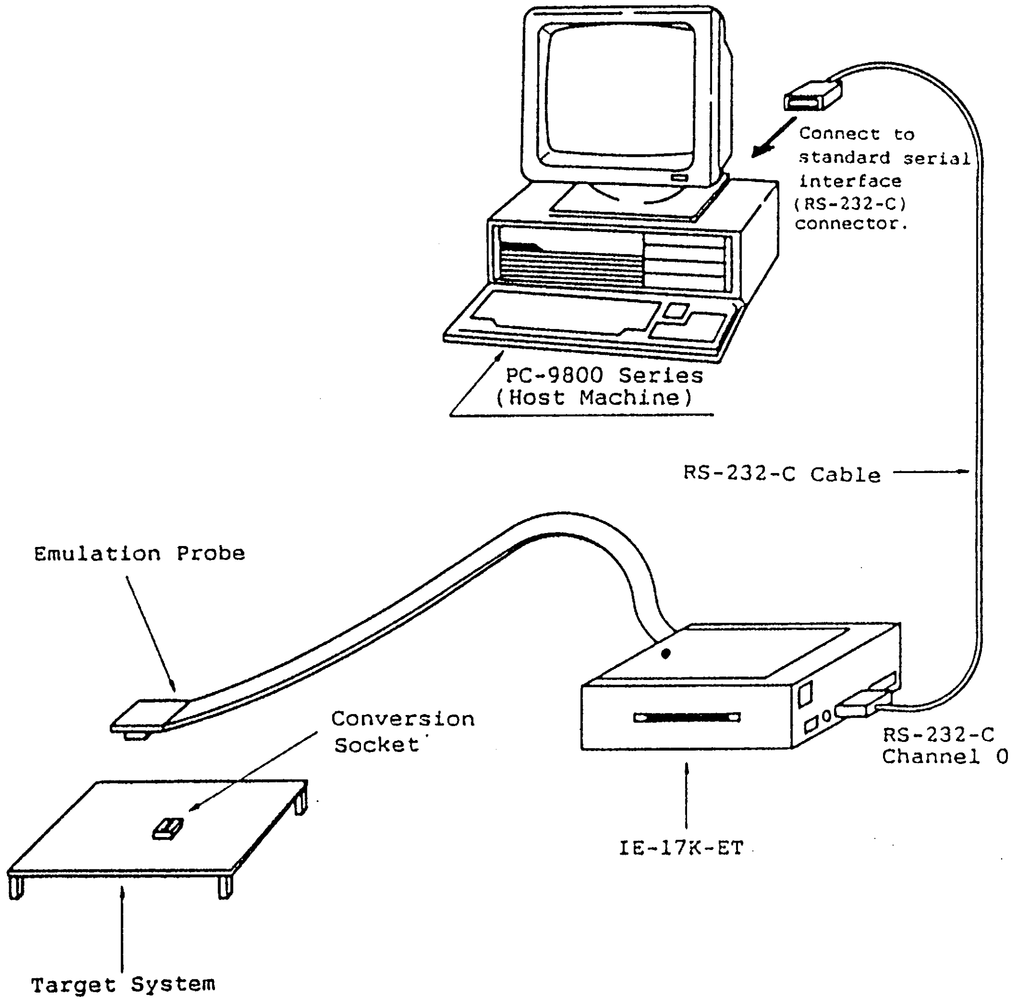

Connect to standard serial interface (RS-232-C) connector.
PC-9800 Series (Host Machine)
RS-232-C Cable
RS-232-C Channel 1
RS-232-C Channel 0
PROM Programmer
IE-17K-ET

3.6 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 3-9 IE-17K-ET and Target System Connection


Connect to
standard serial
interface
(RS-232-C)
connector.
PC-9800 Series
(Host Machine)
RS-232-C Cable
Emulation Probe
Conversion
Socket
RS-232-C
Channel 0
IE-17K-ET
Target System

CHAPTER 4. STARTING

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

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

This chapter introduces the method of communication using a Windows™ (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-ET.

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

Besides having communications functions for use with the IE-17K-ET, 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 if SIMPLEHOST is used.

4.1 COMMUNICATION WITH 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-ET to the PC-9800 series.

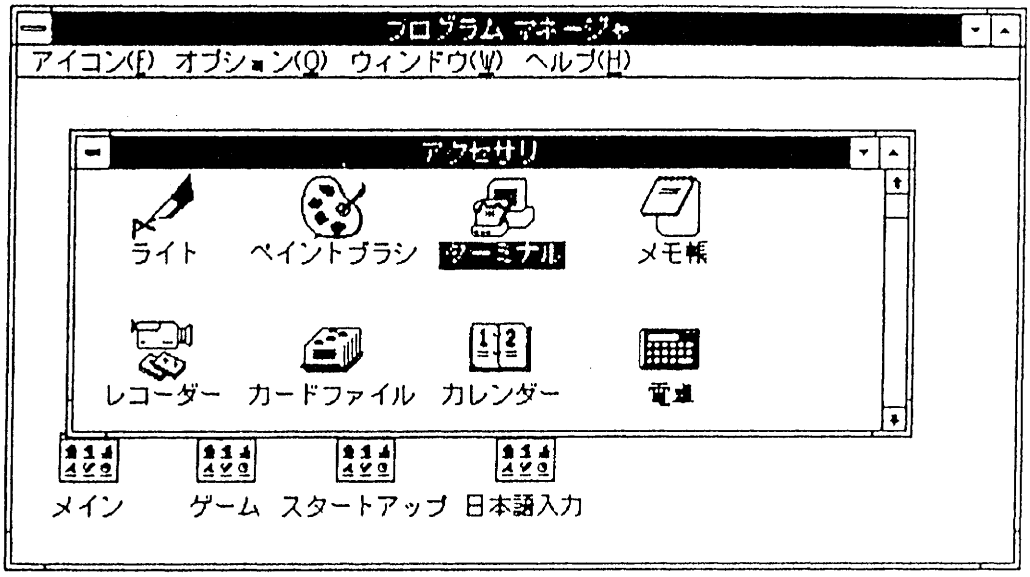
Before starting Windows, check to make sure that the IE-17K-ET in which the SE board is installed and PC-9800 series are connected using the special IE-17K-ET RS-232-C cable, and that the IE-17K-ET power has been turned ON. (Refer to Chapter 3 "Installation".)

4.1.1 TERMINAL START-UP

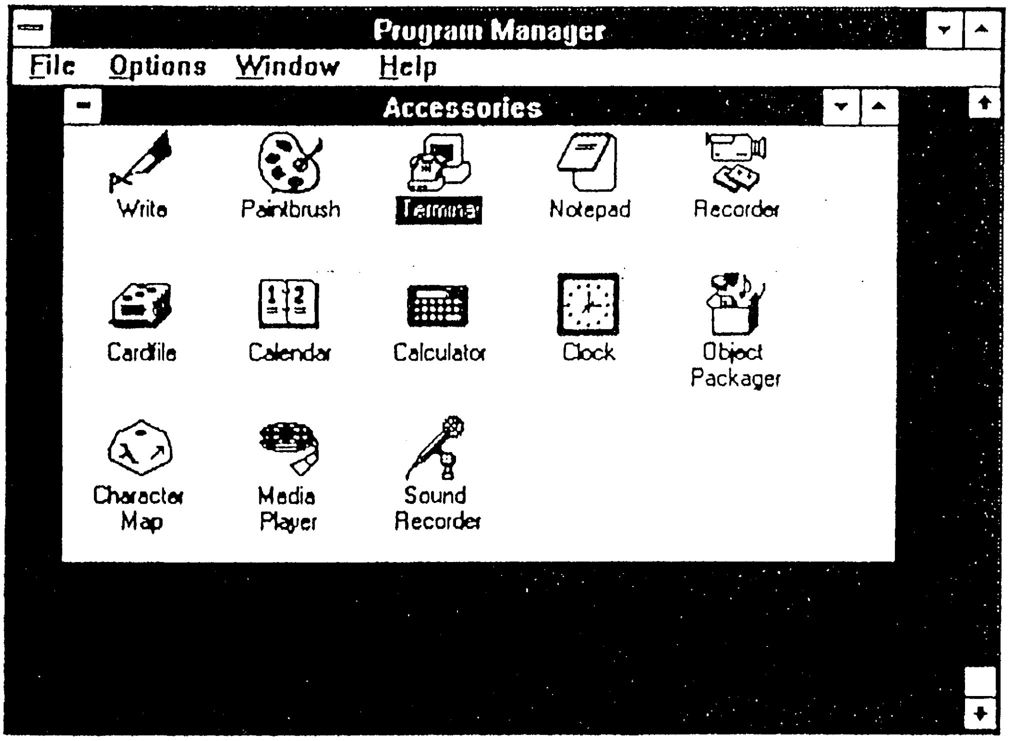
First, starting from the Windows opening screen, select (double click) the "Accessory" group icon, and select the "Terminal" icon, and then execute them (double click). An example selection screen is given below.

Figure 4-1 Terminal Start-up

(a) PC-9800 Series


プログラム マネージャ
アイコン(F) オプション(O) ウィンドウ(W) ヘルプ(H)
アクセサリ
ライト
ペイントブラシ
ターミナル
メモ帳
レコーダー
カードファイル
カレンダー
電卓
メイン
ゲーム
スタートアップ
日本語入力

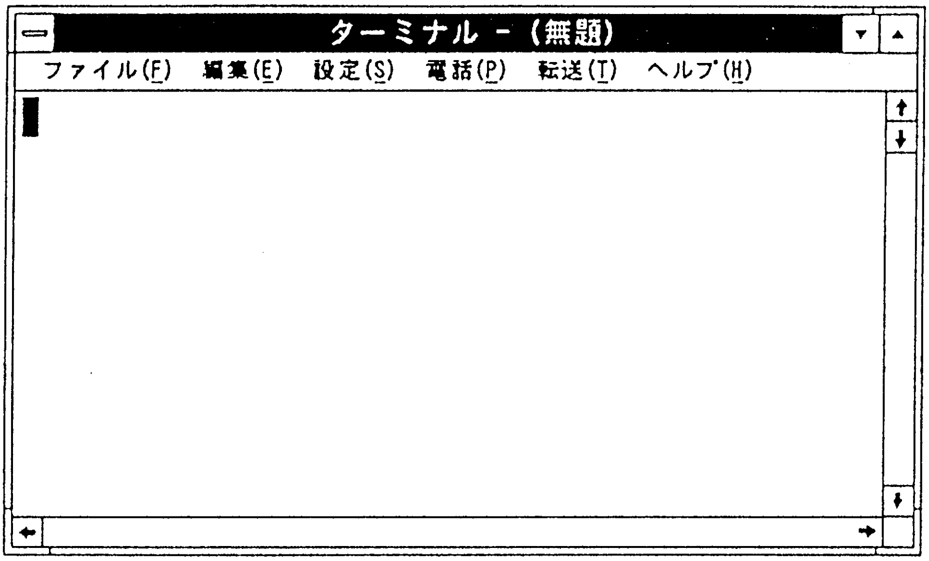
(b) IBM PC/AT


Program Manager
File Options Window Help
Accessories
Write
Paintbrush
Terminal
Notepad
Recorder
Cardfile
Calendar
Calculator
Clock
Object Packager
Character Map
Media Player
Sound Recorder

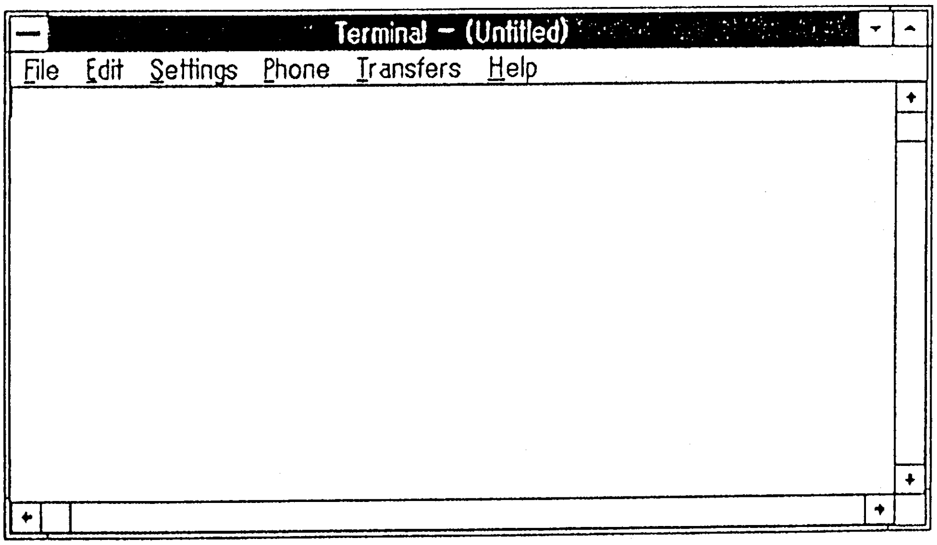
The "Terminal" window is opened after double clicking on the "Terminal" icon of the "Accessory" group.

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

(a) PC-9800 Series


ターミナル - (無題)
ファイル(F)
編集(E)
設定(S)
電話(P)
転送(T)
ヘルプ(H)

(b) IBM PC/AT


Terminal - (Untitled)
File
Edit
Settings
Phone
Transfers
Help

4.1.2 SETTINGS

In order to exchange data and commands with the IE-17K-ET, 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.

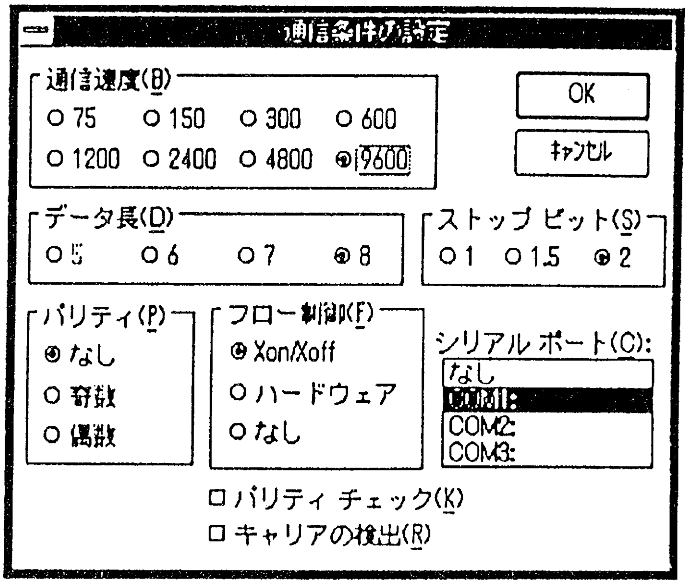
(1) Communication condition settings

The communication conditions for the IE-17K-ET 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 Communications

(a) PC-9800 Series



通信条件の設定

通信速度(B)
○ 75 ○ 150 ○ 300 ○ 600
○ 1200 ○ 2400 ○ 4800 ● 9600

データ長(D)
○ 5 ○ 6 ○ 7 ● 8

ストップ ビット(S)
○ 1 ○ 1.5 ● 2

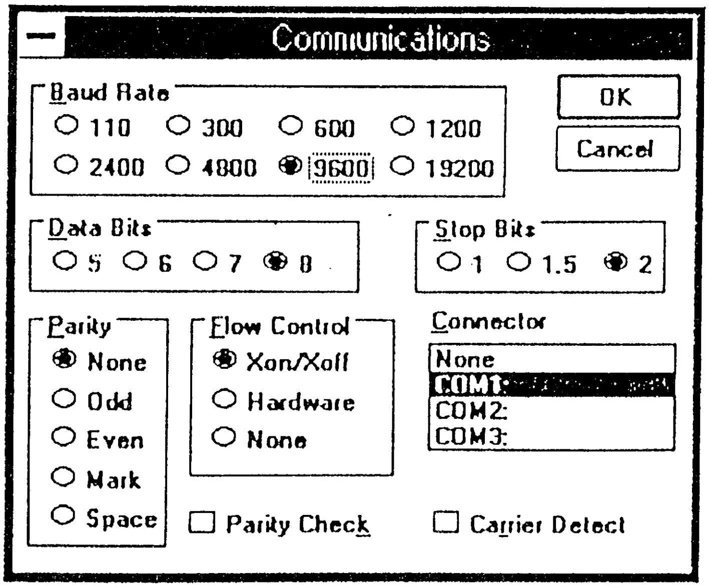
パリティ(P)
● なし
○ 奇数
○ 偶数

フロー制御(F)
● Xon/Xoff
○ ハードウェア
○ なし

シリアル ポート(C):
なし
COM1:
COM2:
COM3:

□ パリティ チェック(K)
□ キャリアの検出(R)

(b) IBM PC/AT



Communications

Baud Rate
○ 110 ○ 300 ○ 600 ○ 1200
○ 2400 ○ 4800 ● 9600 ○ 19200

Data Bits
○ 5 ○ 6 ○ 7 ● 8

Stop Bits
○ 1 ○ 1.5 ● 2

Parity
● None
○ Odd
○ Even
○ Mark
○ Space

Flow Control
● Xon/Xoff
○ Hardware
○ None

Connector
None
COM1:
COM2:
COM3:

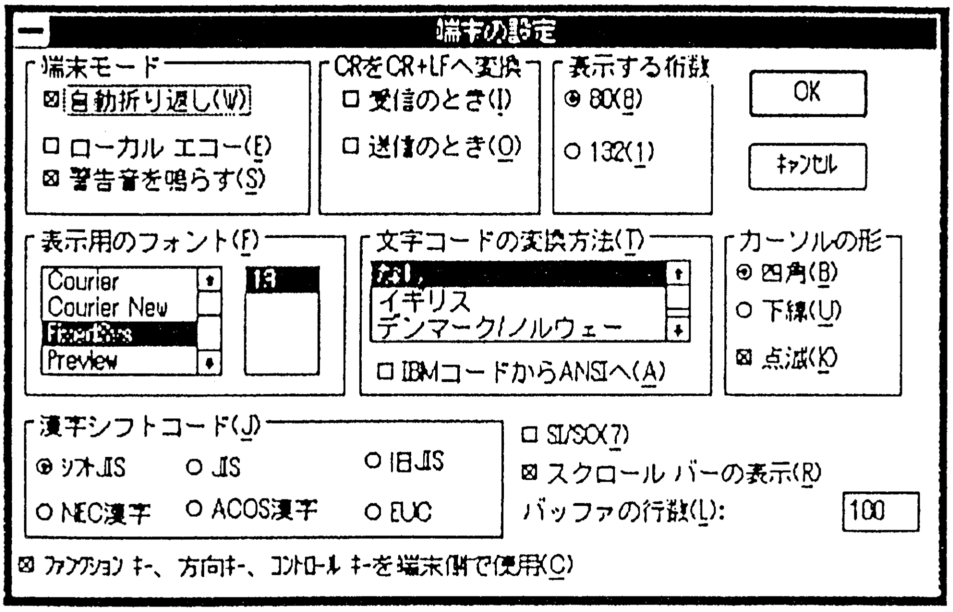
☐ Parity Check
☐ Carrier Detect

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



端末の設定

端末モード
☒ 自動折り返し(W)
☐ ローカル エコー(E)
☒ 警告音を鳴らす(S)

CRをCR+LFへ変換
☐ 受信のとき(I)
☐ 送信のとき(O)

表示する桁数
● 80(8)
○ 132(1)

表示用のフォント(F)
Courier
Courier New
FixedSys
Preview
13

文字コードの変換方法(T)
なし
イギリス
デンマーク/ノルウェー
☐ IBMコードからANSIへ(A)

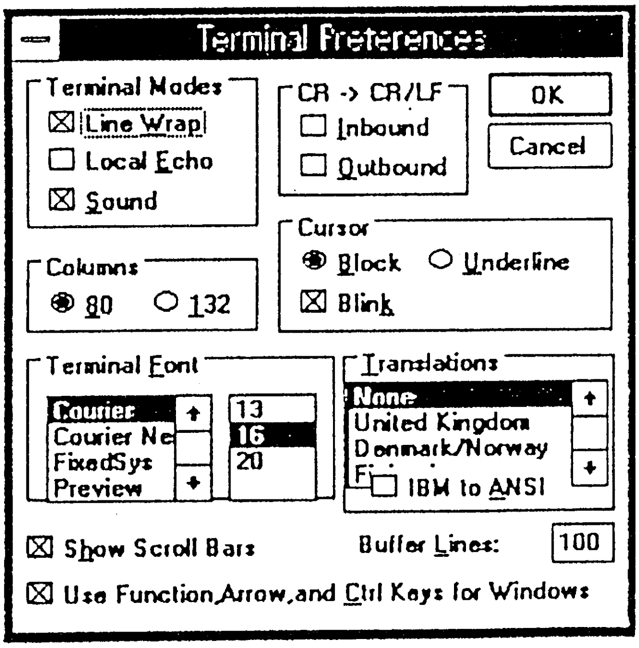
カーソルの形
● 四角(B)
○ 下線(U)
☒ 点滅(K)

漢字シフトコード(J)
● ｼﾌﾄJIS ○ JIS ○ 旧JIS
○ NEC漢字 ○ ACOS漢字 ○ EUC

☐ SI/SO(7)
☒ スクロール バーの表示(R)
バッファの行数(L): 100

☒ ﾌｧﾝｸｼｮﾝ ｷｰ、方向ｷｰ、ｺﾝﾄﾛｰﾙ ｷｰを端末側で使用(C)

(b) IBM PC/AT



Terminal Preferences

Terminal Modes
☒ Line Wrap
☐ Local Echo
☒ Sound

CR -> CR/LF
☐ Inbound
☐ Outbound

Columns
● 80 ○ 132

Cursor
● Block ○ Underline
☒ Blink

Terminal Font
Courier
Courier Ne
FixedSys
Preview
13
16
20

Translations
None
United Kingdom
Denmark/Norway
☐ IBM to ANSI

☒ Show Scroll Bars
Buffer Lines: 100

☒ Use Function,Arrow,and Ctrl Keys for Windows

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

4.1.3 PROGRAM DOWNLOAD TO IE-17K-ET

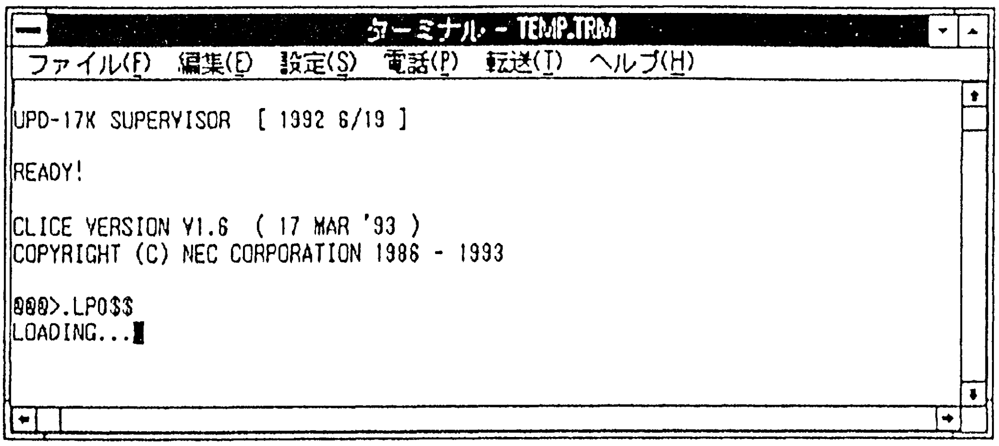
After the "Communications Settings" and "Terminal Preferences" have been completed, it is possible to communicate with the IE-17K-ET. After pressing the reset switch on the "Terminal" window screen, an opening message and @@@> prompt are sent from the IE-17K-ET to the "Terminal" window screen. If the @@@> 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-ET 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-ET to wait for loading. This condition is shown in Figure 4-4.

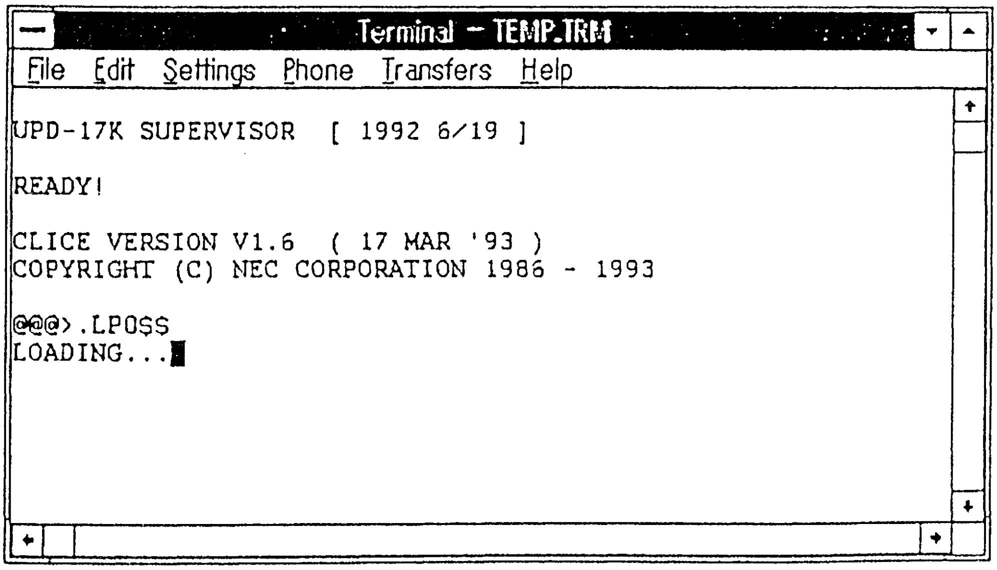
Figure 4-4 Loading Wait Status

(a) PC-9800 Series


ターミナル - TEMP.TRM
ファイル(F) 編集(E) 設定(S) 電話(P) 転送(T) ヘルプ(H)

```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...
```

(b) IBM PC/AT


Terminal - TEMP.TRM
File Edit Settings Phone Transfers Help

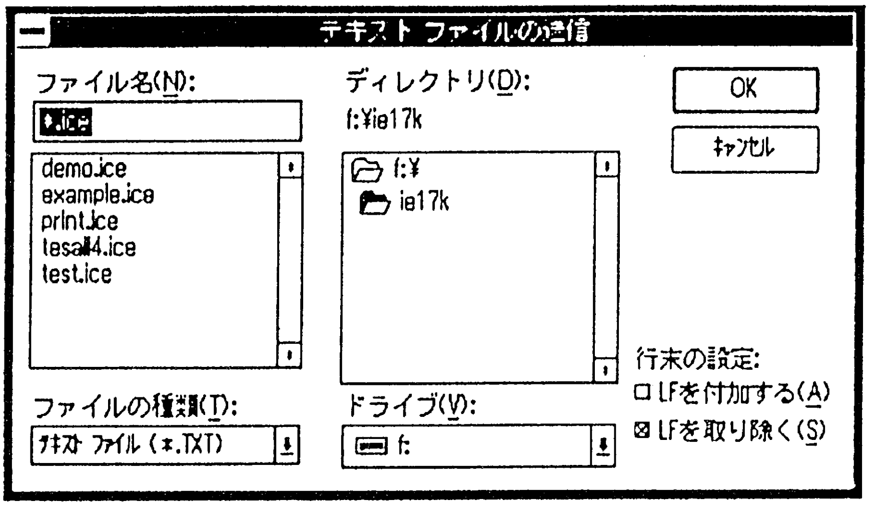
```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...
```

Next, select "Transfers" item from the menu bar in the "Terminal" window, and then select "Text File Transfer". From the "Text File Transfer" screen, input *.ICE in the "File Name". Search for the drive or directory having that file using "Drive" or "Directory", then select the file form 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 transfer of the file.

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

Figure 4-5 Sending Text File

(a) PC-9800 Series



テキスト ファイルの送信
OK
ｷｬﾝｾﾙ

ファイル名(N): *.ice

demo.ice
example.ice
print.ice
tesall4.ice
test.ice

ディレクトリ(D): f:¥ie17k

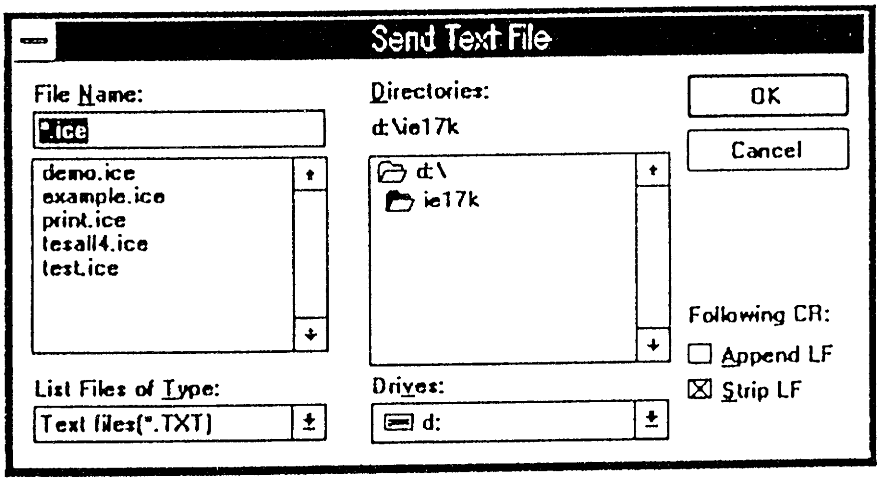
f:¥
ie17k

ファイルの種類(T): ﾃｷｽﾄ ﾌｧｲﾙ (*.TXT)

ドライブ(V): f:

行末の設定:
□ LFを付加する(A)
☒ LFを取り除く(S)

(b) IBM PC/AT



Send Text File
OK
Cancel

File Name: *.ice

demo.ice
example.ice
print.ice
tesall4.ice
test.ice

Directories: d:\ie17k

d:\
ie17k

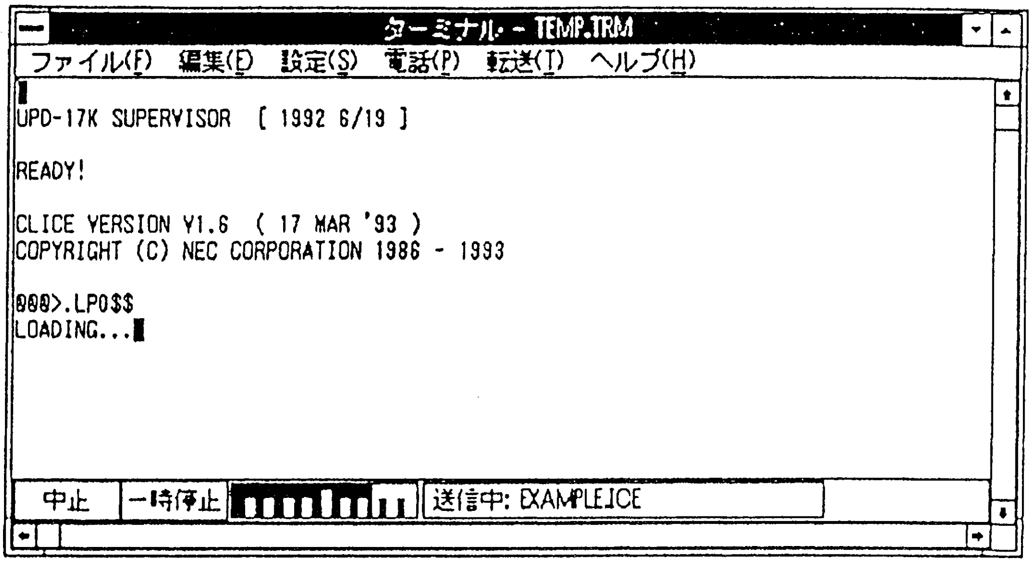
List Files of Type: Text files(*.TXT)

Drives: d:

Following CR:
☐ Append LF
☒ Strip LF

Figure 4-6 File Transfer

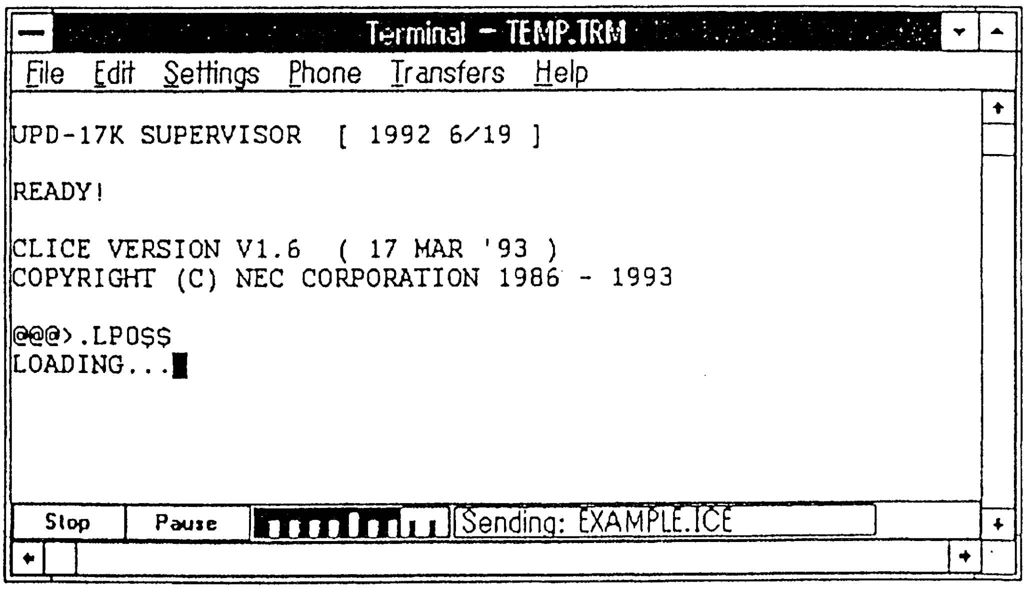
(a) PC-9800 Series


ターミナル - TEMP.TRM
ファイル(F) 編集(E) 設定(S) 電話(P) 転送(T) ヘルプ(H)
中止
一時停止

```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...
```

送信中: EXAMPLE.ICE

(b) IBM PC/AT


Terminal - TEMP.TRM
File Edit Settings Phone Transfers Help
Stop
Pause

```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...
```

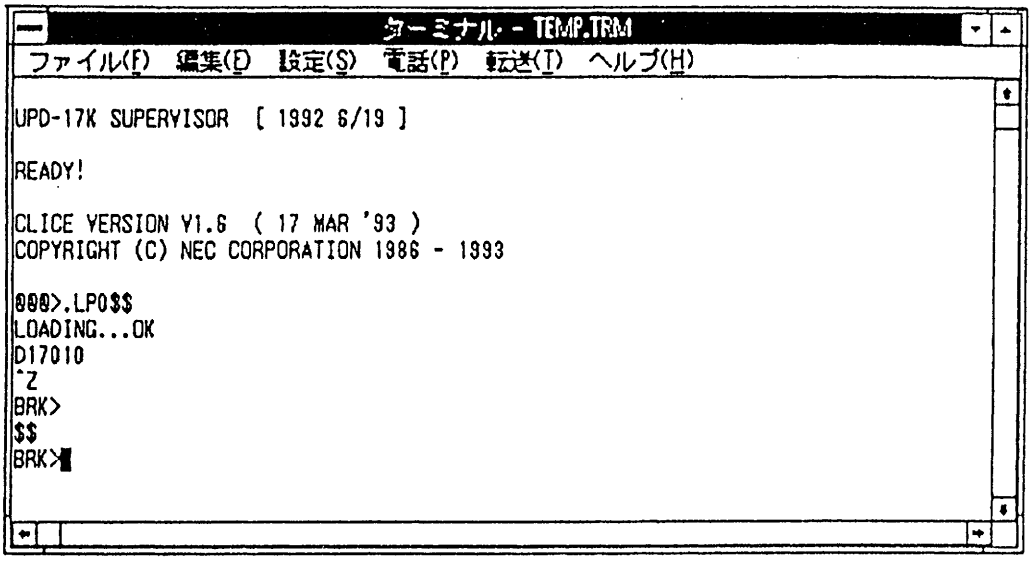
Sending: EXAMPLE.ICE

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 $$, and make sure that the IE-17K-ET is set to be able to receive commands. If the BRK> prompt is returned when the ESC key is pressed twice or when $$ is input, it means that the macro commands built-in to the IE-17K-ET such as .R, or .RN can be used. For details about the built-in macro commands, refer to 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.

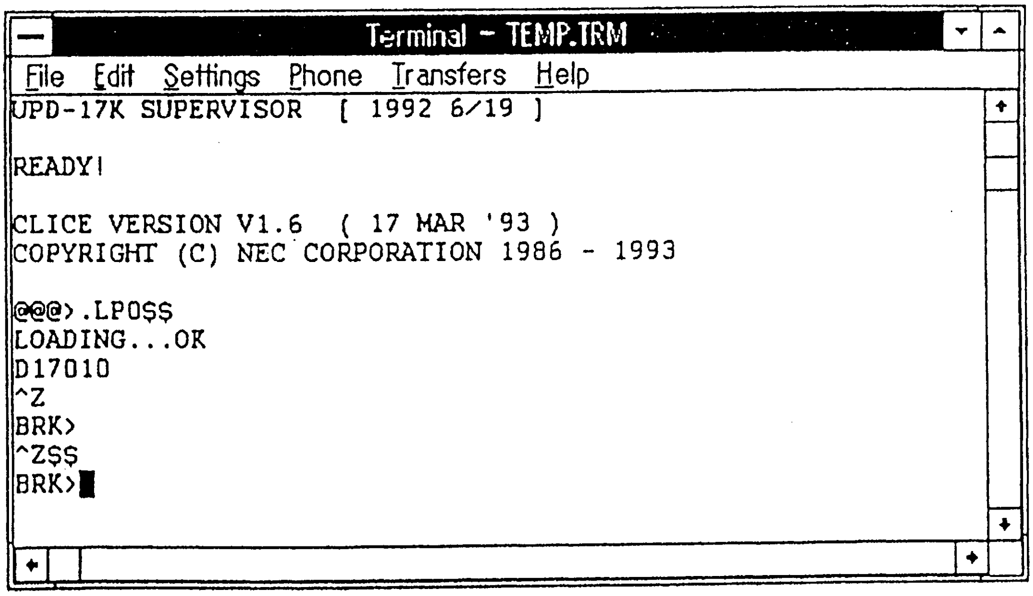
Figure 4-7 File Transfer End

(a) PC-9800 Series


ターミナル - TEMP.TRM
ファイル(F)
編集(E)
設定(S)
電話(P)
転送(T)
ヘルプ(H)

```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...OK
D17010
^Z
BRK>
$$
BRK>█
```

(b) IBM PC/AT


Terminal - TEMP.TRM
File
Edit
Settings
Phone
Transfers
Help

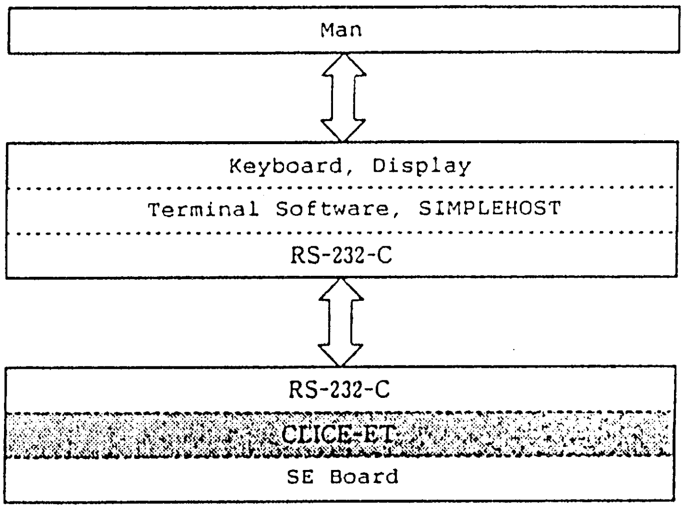
```
UPD-17K SUPERVISOR  [ 1992 6/19 ]

READY!

CLICE VERSION V1.6  ( 17 MAR '93 )
COPYRIGHT (C) NEC CORPORATION 1986 - 1993

@@@>.LP0$$
LOADING...OK
D17010
^Z
BRK>
^Z$$
BRK>█
```

CHAPTER 5. DESCRIPTION OF COMMANDS

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

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

Figure 5-1 CLICE-ET 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-ET.

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-ET (See Appendix A "Primitive Commands").

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

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

READY!

CLICE-ET VER. V*.* (** *** ****)
COPYRIGHT (C) NEC CORPORATION 1986 - 1991

@@@>

*:  Version No. and date
```

5.1 PROMPT

The CLICE-ET prompts show the status of the IE-17K-ET 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-ET 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-ET is started, the model used is not set at the IE-17K, and therefore, the program must be loaded by restarting the IE-17K-ET 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$$ .. 0, 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-ET 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-ET 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-ET 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 Appendix A.5.9 "Others", *command (assignment to Q-register), ? command error display).)

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-ET.

The control characters (ASCII codes 00 to 1FH) from control A (↑A) to control _ (↑^) are displayed by ^ and an alphabetic character.
CLICE-ET 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-ET and the two characters ^ (ASCII code 5EH) and A (ASCII code 41H) are displayed as ^A.

5.4.1 SPECIAL CONTROL CHARACTERS

CLICE-ET 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 (↑C)(ASCII code 03H) is a special control character for interrupt command execution. The two characters ˆ and C are displayed in that order.

When ↑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 Appendix A.5.9 "Others", the *command (assignment to Q-register).)

To interrupt execution after a command is terminated (i.e., after command execution starts), ↑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 (↑X)(ASCII code 18H) key deletes the line containing the cursor.

The command string deleted by ↑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-ET 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-ET 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> ~FFFFFFFC → 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-ET 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

CLIC-ET has arrays and Q-registers as the variable concept.

5.7.1 ARRAY

"Array" represents all the resources managed by CLICE-ET.

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

Therefore, when a value is assigned to an array by CLICE-ET, 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-ET 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-ET 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-ET.

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) Reverse assemble 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-ET 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 HEX file from the RS-232-C channel specified by .LP0 or .LP1.

(Example) Load the program from line 0.

```
@@@>.LP0$$
```

[Notes]

- When the power is turned on, or when the IE-17K-ET is reset (prompt @@@>), load the AS17K HEX 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.

.VP0 .VP1 Verify Program Memory

Format : .VP0 / .VP1

RS-232-C channel 0: VP0
channel 1: VP1

[Function] Verifies the contents of program memory and the data of the AS17K HEX file sent from the RS-232-C channel specified by .VP0 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-ET 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>.VP0$$
Verify...NG
00
BRK>0000000000030303231313011
```

```
: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 (α ≦ β, α > β 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 0, α can be omitted.

When Q-register β is omitted, the code of program memory address α is reverse assembled and displayed. The mnemonic input mode is set. In the mnemonic input mode, it is possible to change the contents of the program memory at mnemonic level.

(Example 1)

```
BRK>5.AP$$
    0005: MOV 05,#5 - $$  $$ input exits the
BRK>                      mnemonic input mode.
BRK>.AP$$
    0000: MOV 00,#1 - MOV 01,#1↵
    0001: MOV 10,#2 - ■

    Waits for input.
```

↵ input performs assembly and moves to the next address. (The BS key can also be used.)

```
BRK>.AP$$
    0000: MOV 00,#1 - MOV 01,#1↵
    ASSEMBLE ERROR
    0000: MOV 00,#1 - ■

    Waits for input.
```

When an error occurs during assembly, it returns to the original address and waits for input.

```
BRK>.AP$$
    0000: MOV 00,#1 - ↵
    0001: MOV 10,#2 - ■

    Waits for input.
```

When ↵ is only pressed, it waits for the next address to be input.

```
BRK>.AP$$
    0000: MOV 00,#1 - MOV 01,#1
    0001: MOV 10,#2 - MOV 01,#1 ■
```

↑P (CTRL + P)

If ↑P are pressed while it is waiting for input, the previously input character string (here that is MOV 01,#1) is displayed.

(Example 2)

```
BRK>USMOV 01,#05
ADD 1,78
$$
BRK>5.APS$$
BRK>5,6.DP$$



0005:  1D015    00781
```

The character string is assigned to the Q register S using the U command.

The contents of Q register S are assembled, and the assembled results are dumped.

[Note] (1) Describe RF addresses by 40H to BFH. However, 00H to 3FH, when written, it treated the same as 80H to BFH. C0H 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 (α ≦ β , α > β : 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 α +3FH.

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

```
BRK>10,20.DP$$

  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 (α ≦ β , α > β : 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 α +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 formal at the RS-232-C channel specified by .XS0 or .XS1.

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

```
BRK>.XS1$$

:1000000063A03CF061273CF0EFE040423CE0EF04AD
:10001000EF10EF20EF30EF91EF00EF10EF20EF3017
:10002000EF90E8C0E8D0E8E0E8F03CA138A538A6B9
:1000300038A738E0E820E8303CF03CF080219030F0
:10004000F7F4601C38E0B204E8E0E8F038A538E0E6
:10005000E830E82038E08031902038E0F6F4605055
```

.Q Restart IE-17K-ET

```
Format :    .Q
```

[Function] Restarts the IE-17K-ET from the previous state when the IE-17K-ET 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-ET 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-ET.

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-ET 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 ($\alpha \leqq \beta$, $\alpha > \beta$: 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 0, α 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
                   instead of ↵
```

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 input.
  0100  3-3  2-■     Waits for next address
                     change input.
```

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-_
```

*: _: Cursor

.DD Dump Data Memory

Format : [α][, β].DD

α : Start address
β : End address (α ≦ β , α > β : Error)

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

When 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 ", β " 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.GEβ

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] [^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 [^A].

When peripheral address 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
```

.PU Put Command

Format : α.PU β

α : Peripheral address
β : Q-register name

[Function] Assigns the contents (numeric) of Q-register β to peripheral address α.

(Example)

BRK>55UA$$	Assign 55H to Q-register A.
BRK>1.PUA$$	Assign the value of Q-register A 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 │ System
        0    0    0    0    0    0    0 ┘  registers
   RP  0123456789ABCDEF
  000  0000000000000320   ..... 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>.S$$
    BR RP PC   INST  MNEMONIC
     0 00 0000 074F0 NOP       :SPC      Advance 1 step at space key input.
     0 00 0001 1D000 MOV 00,#0 :SPC      Advance 1 step at space key input.
     0 00 0002 1D011 MOV 01,#1 :SPC      Advance 1 step at space key input.
     0 00 0003 1000A ADD 00,#A :$$       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 :       │ Execute 4 steps.
     0 00 0002 1D011 MOV 01,#1 :       │
     0 00 0003 1000A ADD 00,#A :       ┘

     |  |   |    |       |
     |  |   |    |   Mnemonic
     |  |   |Instruction
     |  |   |Code
     |  |Program
     |  |Counter
     |Register
     |Pointer
   Bank
```

[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) (however, H) is invalid) is as it 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.

PROG 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 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.

. I) Currently, this item is not supported.

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

```
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 ?     Program Memory
   MATCH(1) / UNMATCH(0) : 0 ?
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ?
   DATA ADDR             : 000 ?
   DATA ADDR MASK        : 000 ?                  Data Memory
   MATCH(1)/UNMATCH(0)-1 : 0 ?
   CURRENT   DATA        : 0 ?
   CURRENT   MASK        : 0 ?
   MATCH(1) / UNMATCH(0) : 0 ?
F) SP LEVEL UPER         : 0 ?
   SP LEVEL LOWER        : 0 ?     Stack Pointer
   MATCH(1) / UNMATCH(0) : 0 ?
J) INTERRUPT ACKNOWLEDGE : 0 ?
   INTERRUPT MASK        : 0 ?     Interrupt
   MATCH(1) / UNMATCH(0) : 0 ?
K) DMA                   : 0 ?
   DMA MASK              : 0 ?   DMA
   MATCH(1) / UNMATCH(0) : 0 ?
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ?
   TERMINAL CONTER     : 0000 ?                                    Counter
   COUNTER  MASK       : 0000 ?
   MATCH(1) / UNMATCH(0) : 0 ?
```

<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 ? │ Program Memory
   MATCH(1) / UNMATCH(0) : 0 ? ┘
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : ? ┐
   DATA ADDR            : 000 ?                   │
   DATA ADDR MASK       : 000 ?                   │
   MATCH(1) / UNMATCH(0) : 0 ?                    │
   CURRENT   DATA        : 0 ?                    │ Data Memory
   CURRENT   MASK        : 0 ?                    │
   MATCH(1) / UNMATCH(0) : 0 ?                    │
   PREVIOUS DATA DISABLE YES(1) / NO(0) : 0 ?     │
   PREVIOUS DATA         : 0 ?                    │
   MATCH(1) / UNMATCH(0) : 0 ?                    ┘
I) RELEASE MAR FROM AND YES(1) / NO(0) : 0 ? ┐    *
   MAR   DATA            : 0 ?               │ MAR
   MAR   MASK            : 0 ?               │
   MATCH(1) / UNMATCH(0) : 0 ?               ┘
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ? ┐
   TERMINAL CONTER   :   00 ?                                     │ Counter
   COUNTER  MASK     :   00 ?                                     │
   MATCH(1) / UNMATCH(0) : 0 ?                                    ┘
```

*: 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 ?  │ Program Memory
   MATCH(1) / UNMATCH(0) : 0 ?  ┘
E) RELEASE DATAMEMORY FROM AND YES(1) / NO(0) : 0 ? ┐
   DATA ADDR           : 000 ?                      │
   MATCH(1) / UNMATCH(0) : 0 ?                      │ Data Memory
   CURRENT   DATA        : 0 ?                      │
   MATCH(1) / UNMATCH(0) : 0 ?                      ┘
L) COUNTER SOURCE SELECT
   NO(0) / INST(1) / CONDITION(2) / INST AFTER CONDITION(3) : 0 ? ┐
   TERMINAL CONTER    :   00 ?                                    │ Counter
   COUNTER  MASK      :   00 ?                                    │
   MATCH(1) / UNMATCH(0) : 0 ?                                    ┘
```

<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 ?  │ Program Memory
   MATCH(1) / UNMATCH(0) : 0 ?  ┘
G) INST CODE        : 00000 ?   ┐
   INST MASK        : 00000 ?   │ Instruction Code
   MATCH(1) / UNMATCH(0) : 0 ?  ┘
```

<Break condition setting items (for level 2)>

Level 2 is used when setting each unit (unit 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
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 ? TUSS
```

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.

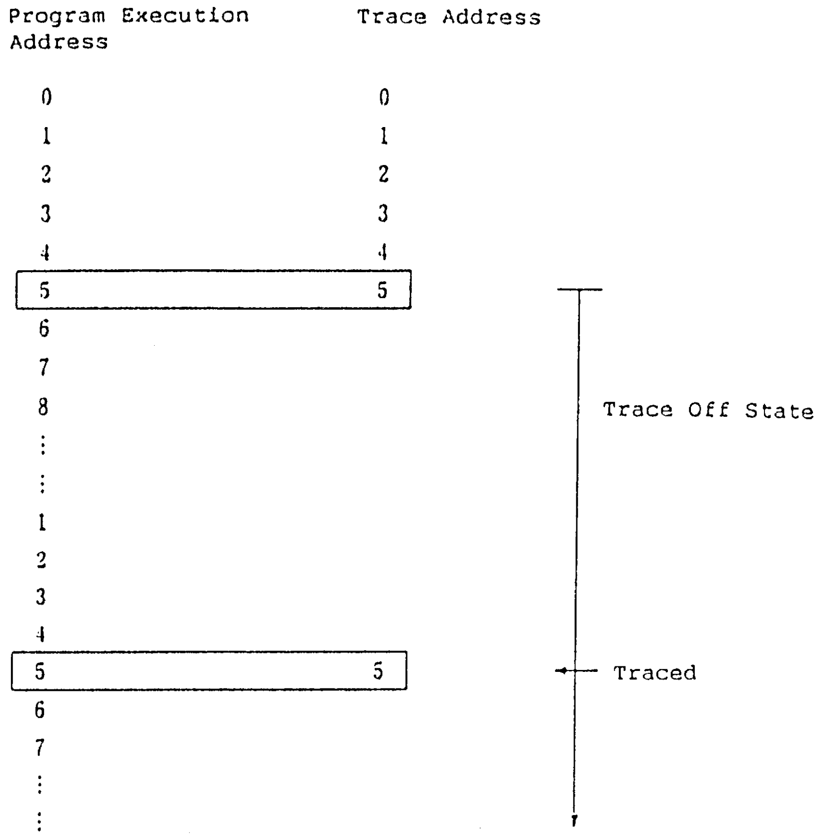
- After TRACE ON and TRACE OFF, that state is held even if set to TRACE DON'T CARE.

  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.


Program Execution Address
Trace Address
0 0
1 1
2 2
3 3
4 4
5 5
6
7
8
⋮
⋮
1
2
3
4
5 5
6
7
⋮
⋮
Trace Off State
Traced

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

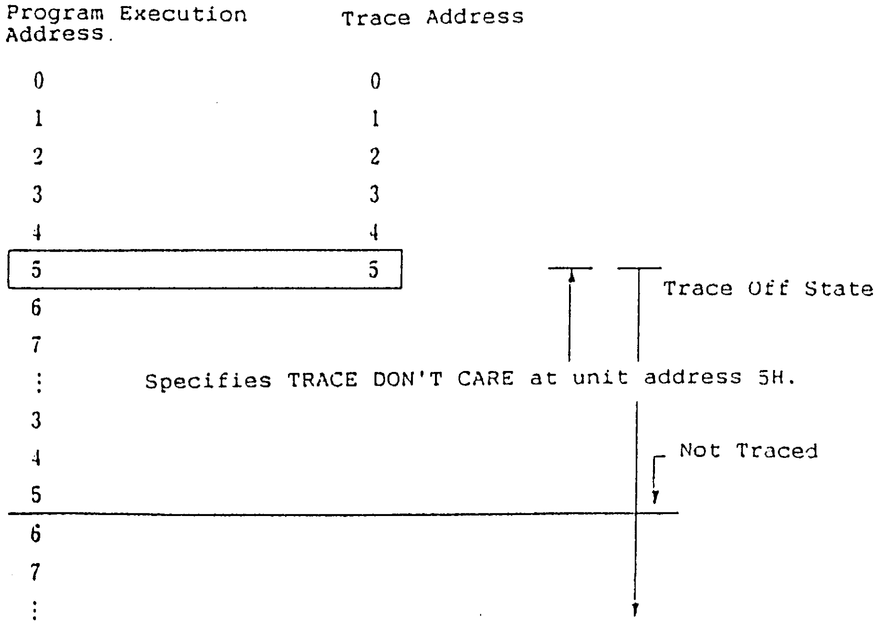

Program Execution Address.
Trace Address
0 0
1 1
2 2
3 3
4 4
5 5
6
7
⋮
3
4
5
6
7
⋮
Trace Off State
Specifies TRACE DON'T CARE at unit address 5H.
Not Traced

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

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

```
BRK>0,10.DT$$
ADDRESS (1) / STATUS (0) TRACE ? 0 ↵

TR_NO      ADDR INSTRUCTION          WA  DB JG  TIME
0000 0000 0000 MOV 00  ,#A 1D00A 000  A  0  0000001
0001 0001 0001 MOV 01  ,#B 1D01B 001  B  0  0000002
0002 0002 0002 MOV 02  ,#C 1D02C 002  C  0  0000003
0003 0003 0003 MOV 03  ,#D 1D03D 003  D  0  0000004
0004 0004 0004 MOV 04  ,#E 1D04E 004  E  0  0000005
0005 0005 0005 MOV 05  ,#F 1D05F 005  F  0  0000006
0006 0006 0006 MOV 06  ,#0 1D060 006  0  0  0000007
```

```
0007 0007 0007 MOV 07  ,#1 1D071 007 1 0 0000008
0008 0008 0008 MOV 08  ,#2 1D082 008 2 0 0000009
0009 0009 0009 MOV 09  ,#3 1D093 009 3 0 0000010
0010 000A 000A NOP         074F0 04F 0 0 0000011
0011 000B 000B NOP         074F0 04F 0 0 0000012
0012 000C 000C NOP         074F0 04F 0 0 0000013
0013 000D 000D NOP         074F0 04F 0 0 0000014
0014 000E 000E NOP         074F0 04F 0 0 0000015
0015 000F 000F BR   000F   0C00F 000 F 0 0000016
0016 0010 000F BR   000F   0C00F 000 F 0 0000017
(1)  (2)  (3)     (4)       (5)  (6)(7)(8) (9)
```

(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) Data memory write address
Effective when data is written to data memory.*

(7) Data bus
Shows the value written when data is written to data memory.*

(8) * is displayed for instructions that are skipped at skip instruction execution.

(9) 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 IE-17K-ET 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$$
:104143000B0B0B0B00FF020202000000000001003A
:1041530000010100000000FFFF0000100000FF0746
:104163000B000F0400000F00040000FFFF00C0011C
:10417300000001000F00000101000101000000FF29
:10418300FF000010000000FFFF0000100000FF0709
:104193000B000F0400040F00000000FFFF000001EC
:1041A300000001000F04000100000100000000FFF7
:1041B300FF000008000000FFFF0000100000FF07E1
:1041C3000B000F0400000F00000000FFFF000001C0
:1041D300000001000F00000100000100000000FFCB
:1041E300FF000008000000FFFF0000100000FF07B1
:1041F30000000F0000000F00000000FFFF1000018F
:10420300000001000F00000100000100000000FF9A
:07421300FF000000000000A5
:00000001FF
```

.LC0 .LC1 Load Break/Trace Condition

$$\text{Format : } \left\{ \begin{array}{l} \text{.LC0} \\ \text{.LC1} \end{array} \right\}$$

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

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

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

```
BRK>.LC0$$
```

.VC0 .VC1 Verify Break/Trace Condition

Format : $\left\{\begin{matrix}\text{.VC0}\\\text{.VC1}\end{matrix}\right\}$

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

[Function] Verifies the break/trace conditions in the IE-17K-ET and the data sent to the IE-17K-ET specified by .VC0 or .VC1.

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 line 0.

```
BRK>.VC0$$
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 (α ≦ β ;
β :	End address α > β : 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>.DM$$
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 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

(Example 2) Display the contents of DATA coverage.

```
BRK>.DM$$
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 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 .SP0 .SP1 .LP0 .LP1 .VP0 .VP1 .XS0 XS1
                   << PROGRAM MEMORY COMMAND  >>
.ID .CD .DD .D
                   << DATA MEMORY COMMAND  >>
.R  .RN .BG .BK .CA .S
                   << EMULATION COMMAND  >>
.CC .CT .DC .DT .DM .SC0 .SC1 .LC0 .LC1 .VC0 .VC1
                   << BREAK , TRACE CONDITION COMMAND  >>
```

CHAPTER 6. PROGRAM EXECUTION

The following program execution methods are available.

(1) Real-time emulation

(2) 1-step emulation

6.1 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>
```

6.2 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 6-1.)

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 6-2.)

Figure 6-1 Break Condition Setting

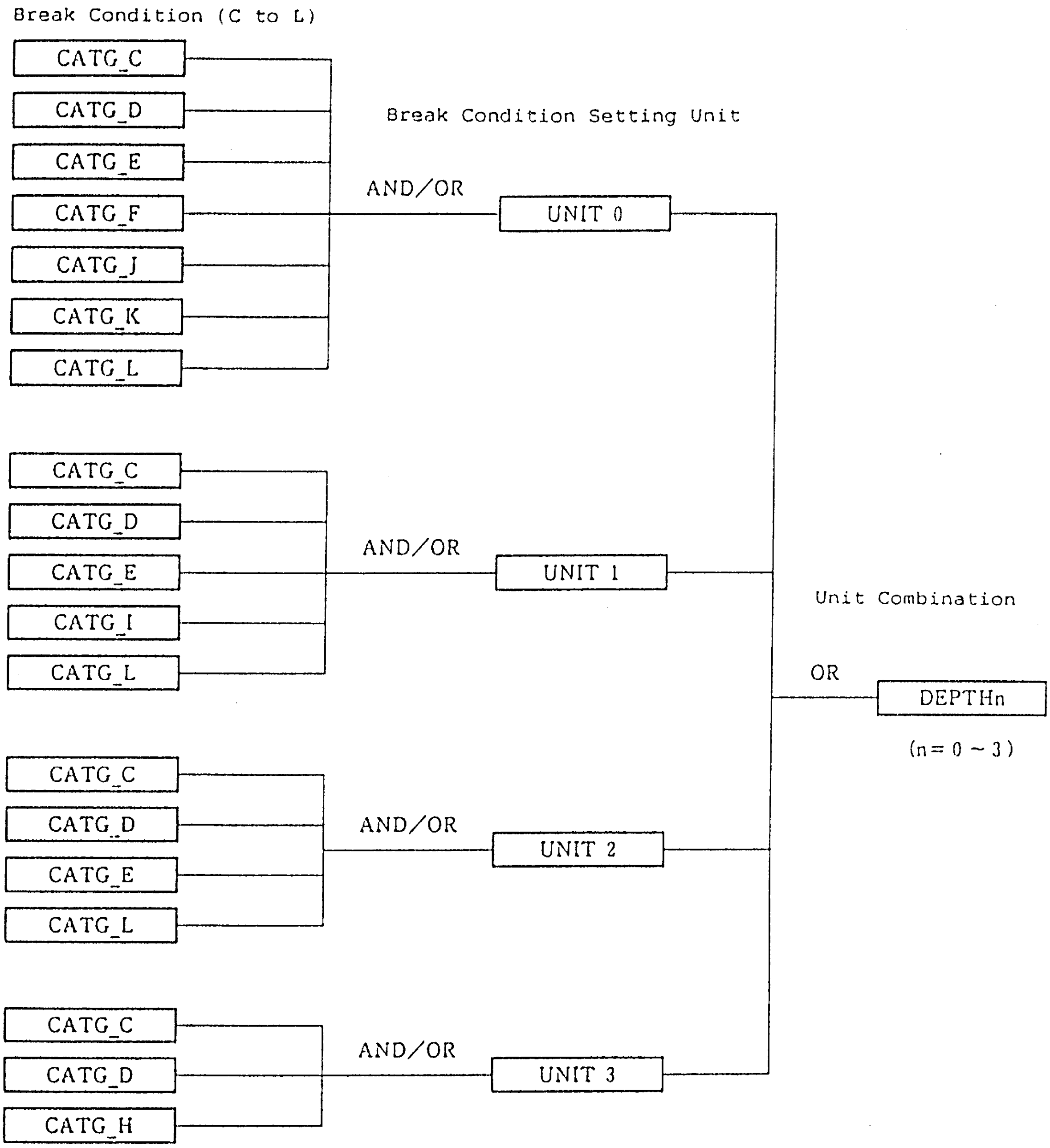
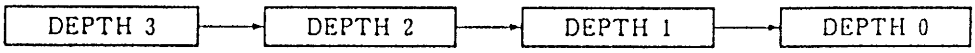

Break Condition (C to L)
CATG_C
CATG_D
CATG_E
CATG_F
CATG_J
CATG_K
CATG_L
Break Condition Setting Unit
AND/OR
UNIT 0
CATG_C
CATG_D
CATG_E
CATG_I
CATG_L
AND/OR
UNIT 1
Unit Combination
OR
DEPTHn
(n= 0 ~ 3)
CATG_C
CATG_D
CATG_E
CATG_L
AND/OR
UNIT 2
CATG_C
CATG_D
CATG_H
AND/OR
UNIT 3

Figure 6-2 Break by Break Condition Sequence


DEPTH 3
DEPTH 2
DEPTH 1
DEPTH 0

6.2.1 BREAK BY PROGRAM ADDRESS

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

(Example 1) Break when the program address reached OOFOH.

```
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) : ? 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 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) 7 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) / 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          Break when condition above
   COUNTER MASK       : 00 ? FF         establishes 5 times
   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 ? 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>
```

6.2.2 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 the
   CURRENT DATA         : 0 ?  0         address above

   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
   MATCH(1)/UNMATCH(0)  : 0 ? 1          before 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
00F0 1D309   BREAK
00F1 1D059   OVERRUN
00F2 1D045   NEXT
 PC  SP   AR  WR BR  MP  IX
00F2  0  0000  0  1  000 000
PSW :DB  CP  CY   Z IXE MPE JG
      0   0   0   0  0   0   0
RP   0123456789ABCDEF
00   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
   CURRENT MASK          : 0 ?  F        value is 5H
   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
00F0 1D305    BREAK
00F1 1D059    OVERRUN
00F2 1D045    NEXT
 PC  SP   AR   WR BR  MP  IX
00F2  0  0000  0  1  000 000
PSW :DB  CP  CY   Z IXE MPE JG
      0   0   0   0  0   0   0
RP   0123456789ABCDEF
00   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 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
00F0 1D309    BREAK
00F1 1D059    OVERRUN
00F2 1D045    NEXT
 PC   SP    AR   WR BR   MP   IX
00F2  0   0000  0   1   000 000
PSW :DB  CP   CY    Z  IXE MPE JG
      0   0    0    0  0    0   0
RP    0123456789ABCDEF
00    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
                                           specification
   MATCH(1)/UNMATCH(0) : 0 ?  1
   CURRENT DATA           : 0 ?  5         Break when write
                                           value is 5H
   CURRENT MASK           : 0 ?  F
   MATCH(1)/UNMATCH(0) : 0 ?  1
   PREVIOUS DATA DISABLE YES(1)/NO(0) : 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
00F0 1D305    BREAK
00F1 1D059    OVERRUN
00F2 1D045    NEXT
 PC   SP    AR   WR  BR   MP   IX
00F2  0   0000   0   1   000 000
PSW :DB   CP   CY    Z  IXE  MPE  JG
      0    0    0    0   0    0    0
RP   0123456789ABCDEF
00   0000099990008005
BRK>
```

6.2.3 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      Break when unit 0 or unit 1
   INITIAL DEPTH : 0 ? 0               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) 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 ? 1000    To DEPTH0 condition when unit
     DEPTH-0       : 1101 ? 0100    0 establishes
   INITIAL DEPTH : 0 ? 1            Break when unit 1 establishes

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>
```

6.3 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   _          ......  When a space is input, the
 * *0 003D 11001   $$                 program is executed 1  step

BRK>
```

CHAPTER 7. SE BOARD PROM CREATION

Programs corrected by IE-17K-ET 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.

CHAPTER 8. ERROR MESSAGES

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

8.1 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) ?ILL ILLEGAL COMMAND

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

Use the correct command.

(7) ?INA ILLEGAL NUMBER OF ARGUMENTS

Number of macro command arguments is small.

Correct input the macro command arguments.

(8) ?IPE INPUT ERROR

Illegal value is set by .CC command.

Reset the correct value.

(9) ?IQN ILLEGAL Q-REGISTER NAME

Q-register name is incorrect.

Use the correct Q-register name.

(10) ?IVA INVALID ARGUMENT

An illegal value is specified at an argument.

Specify the correct argument.

(11) ?IWT ILLEGAL WAIT TIME VALUE

Z command argument is incorrect.

Specify the correct argument.

(12) ?MLA MISSING<

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

Correct the < > correspondence.

(13) ?MLP MISSING)

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

Correct the ( ) correspondence.

(14) ?MNF MACRO COMMAND NOT FOUND

Character string beginning with '.' is not found as a built-in macro command.

Input the correct built-in macro command.

(15) ?MRA MISSING>

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

Correct the <   > correspondence.

(16) ?MRP MISSING(

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

Correct the (   ) correspondence.

(17) ?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.

(18) ?NAE NO ARGUMENT BEFORE=

No = command argument.

Input the argument.

(19) ?NAQ NO ARGUMENT BEFORE"

No " command argument.

Input the argument.

(20) ?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.

(21) ?POS INVALID ADDRESS

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

Specify a value within the last program memory address.

(22) ?QNF QUOTATION NOT FOUND

' not found.

Input the '.

(23) ?QRO Q-REGISTER AREA OVERFLOW

Q-register area is full.

Delete the contents of the unwanted Q-register.

(24) ?QST Q-STACK OVERFLOW

Q-stack area is full.

Delete the unwanted contents of the Q-register.

(25) ?QUN Q-STACK UNDERFLOW

There is no value at the Q-stack area.

Assign at value to the Q-stack area.

(26) ?RNE CPU RUN ERROR

An unexecutable command is input during program execution.

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

(27) ?RNG NUMERIC RANGE OVER

Numeric exceeds the effective range.

Set the numeric to within the effective range.

(28) ?RSE CPU RESET ERROR

An unexecutable command is input during program execution.

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

(29) ?RTE CPU RESET ERROR

An unexecutable command is input during program execution.

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

(30) ?SYN INVALID SYNTAX

Syntax is incorrect.

Use the correct syntax.

(31) ?TAG MISSING TAG !XXX!

Tag is not found.

Specify the correct tag.

(32) ?UPE UNPRINTABLE ERROR

An error that is not available with the CLICE-ET system is generated. Restart it.

(33) ?UPO UNTERMINATED POINTER @ OR _

@ command or _ command not terminated by | V.

Terminate with | V.

(34) ?UTG UNTERMINATED TAG

Tag not enclosed in !.

Enclose tag in !.

(35) ?WRE WRITE ERROR

When writing to memory, a verify error is generated.

Execute the command again.

8.2 HARDWARE ERRORS

Hardware errors are error messages that are displayed when the IE-17K-ET 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

Displays 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

Displays when the option switch specification described when a program is loaded to the IE-17K-ET 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-ET 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-ET 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 left 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-ET.

(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 (15) below are displayed when there is an error in the result of IE-17K-ET 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-ET.

(11) DEVICE ERROR → PTC (UPD71054) #0

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

(12) DEVICE ERROR → PTC (UPD71054) #1

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

(13) DEVICE ERROR → SCU (UPD71051) #0

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

(14) DEVICE ERROR → SCU (UPD71051) #1

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

(15) DEVICE ERROR → ICU (UPD71059)

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

Error messages (16) to (21) below are displayed when IE-17K-ET CPU runs away or otherwise malfunctions.

(16) <<DIVIDE BY ZERO>>

Divided by '0'.

(17) <<CHECK FIELD>>

Memory boundary crossed.

(18) <<SINGLE STEP>>

Single step is performed.

(19) <<BREAK MODE>>

Break instruction is executed.

(20) <<OVERFLOW>>

Overflow is generated during operation.

(21) <<NMI>>

NMI is generated.

APPENDIX A. PRIMITIVE COMMANDS

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

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-ET 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:TM0, 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
CRG	Condition register top address
CR0	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
RNI	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 ☐	System register PC address
SP ☐	System register SP address
AR0	System register AR0 address
AR1	System register AR1 address
AR2	System register AR2 address
AR3	System register AR3 address
WR ☐	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 ☐	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)

(to be continued)

Table A-2 Array Table (cont'd)

Array Name	Function
DSW	DIP SWITCH
RES	RESET Mam'Chip
SEN	SE BOARD NUMBER

Remarks: indicates 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 ↑FCRG+ ↑FDP0=H (address that stores the break condition DEPTH0 in hexadecimal), etc.

Table A-3 Condition Register Offset Address

Array Name	Function
DP0	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
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 ↑FCR0+ ↑FPAU=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
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
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 DESCRIPTION OF PRIMITIVE COMMANDS

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-ET.

New commands (macro commands) can be added to the IE-17K-ET 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 : _ α↑V

α : 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-ET has two kinds of functions, "built-in functions" which are built into the system and "user-defined functions" (↑Q commands) which are defined by the user.

Various built-in functions, including array address function (↑F command), key input function (↑T command), and numeric function (¥ command), are available.

(1) User-defined function
↑Q

(2) Array address function
↑F

(3) Key input function
↑T

(4) Numeric function
¥

↑Q User-defined Function

Format : ↑Qα

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 Functions

Format : ↑Fα

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) ↑TU1↑TU2↑TU3
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-ET 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 βUα
② Uαγ$ (macro definition)
③ δ, εUα

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.

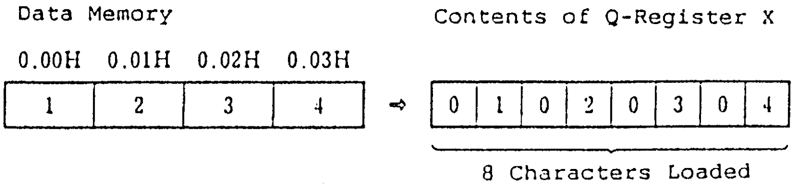
When a character string is assigned as character string γ, it can be used as a macro.

(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) ↑FDTM, ↑FDTM+3UX

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


Data Memory
0.00H
0.01H
0.02H
0.03H
1
2
3
4
Contents of Q-Register X
0
1
0
2
0
3
0
4
8 Characters Loaded

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, ↑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.

```
↑FWRKUP  < ↑TU2      D-Q2   ; Q2, QPXB           %P>
QP=^FWRK   Q2=key    if CR ?  ARRAY (Q1)=Q2     QP=QP+1
```

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-bit 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, ↑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, ↑FDTMXBQ1, ↑FDTM+1XB

XC Character String Assignment to Array

Format : βXCα

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) $\underset{①}{\underline{\text{U1ABC\$}}}$ $\underset{②}{\underline{\uparrow\text{FWRKXC1}}}$

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 ↑FWRK, B is assigned to ↑FWRK+1, and C is assigned to ↑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-ET 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 (↑Q command).

(1) Macro command execution
M

(2) User function execution
↑Q

M Macro Command Execution

Format : Mα

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 : ↑Qα

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-register 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-ET 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 0 command.

The character string enclosed in ! has no affect on execution of the command.

(Example) `↑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 : $\alpha"\beta\gamma'\delta$
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 ", 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 ($n \neq 0$)
L: Less than zero ($n<0$)
G: Greater than zero ($n>0$)

(Example) `Q1-3" E0U2'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, 0U2 is not skipped and Q1+Q2U1 following ' is executed.

A.5.8 DISPLAY

CLICE-ET 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 OAH. 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 OAH 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) ↑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 :	$=C\alpha$
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) ↑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-ET 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) `↑D1000Z`

Halts execution for 10 seconds.

* Assignment to Q-register

Format : $*\alpha$

α : Q-register name

[Function] Assigns the contents of the command buffer to Q-register α.

When command string input is interrupted by ↑C and when command execution is interrupted by ↑C↑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_ ↑FPRM+ (Q1 {1) ↑V=H$ ↑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 ↑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.

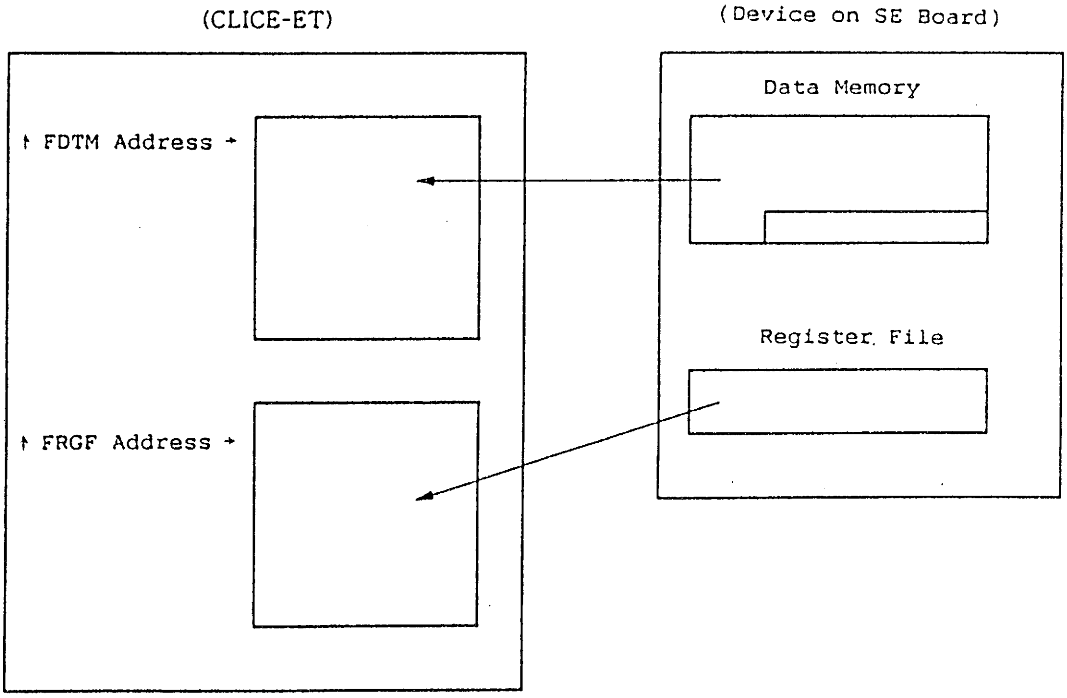
This function allows efficient command input without the need to re-input the command from the beginning even when a long command string is input and an error is generated.

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-ET.

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.

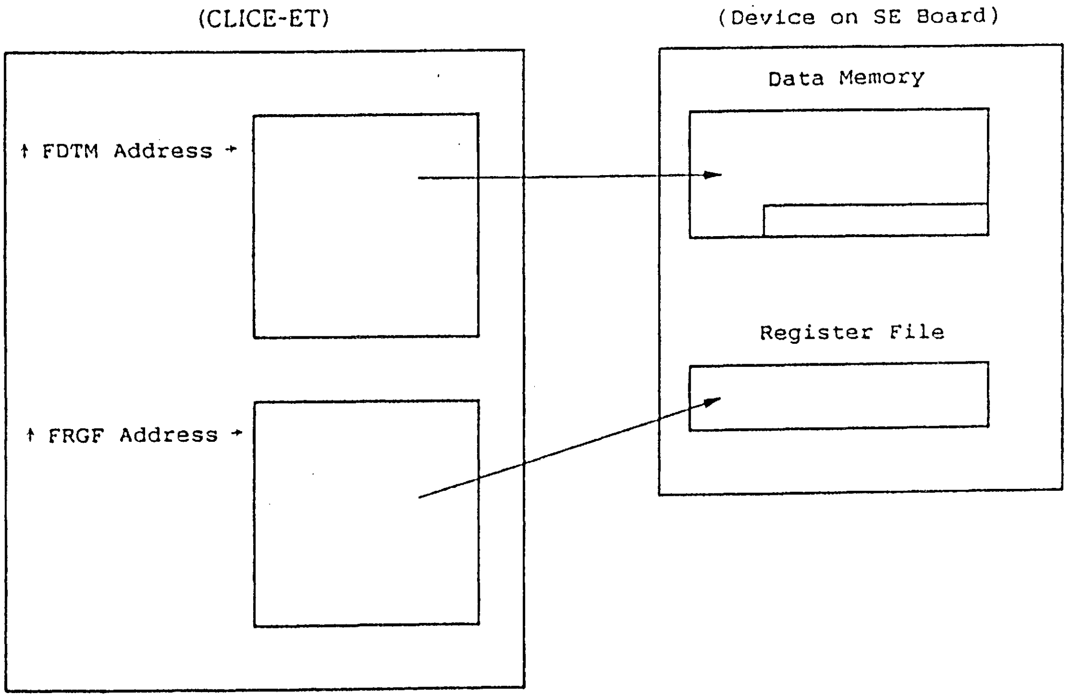

(CLICE-ET)
(Device on SE Board)
↑ FDTM Address →
↑ FRGF Address →
Data Memory
Register File

W Data Memory Write Command

Format : W

[Function] Writes the data on CLICE-ET 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.


(CLICE-ET)
(Device on SE Board)
↑ FDTM Address →
↑ FRGF Address →
Data Memory
Register File

A.6 EDITOR

CLICE-ET 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-ET 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-ET 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¥E¥■	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 I.

>ABCDEFG■
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)	>ABCDEFG■	Original character string.
	>ABC■	Cursor is moved.
	>ABCDEFG	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

The built-in macro commands have been gathered in table format. This table should be helpful during program development. This list can also be used as an index.

The IE-17K-ET does not support the PPG (program pattern generator) and so there are no PPG control commands.

B.1 PROGRAM MEMORY CONTROL COMMANDS

Command Name	Function	Summary	Page
.LP0	Program memory load (channel 0)	Downloads the xxx.ICE file using channel 0 of the RS-232-C after the IE-17K-ET has started up.	5-22
.LP1	Program memory load (channel 1)	Downloads the xxx.ICE file using channel 1 of the RS-232-C after the IE-17K-ET has started up.	5-22
.VP0	Program memory verify (channel 0)	Uses channel 0 of the RS-232-C to verify the program memory with the ICE file downloaded to the IE-17K-ET.	5-23
.VP1	Program memory verify (channel 1)	Uses channel 1 of the RS-232-C to verify the program memory with the ICE file downloaded to the IE-17K-ET.	5-23
.IP	Program memory initialization	Rewrites a selected range of the IE-17K-ET program area with specified data (1-4-3-4-4 format).	5-25
.CP	Program memory change	Rewrites a selected address of the IE-17K-ET program area for each instruction (1-4-3-4-4 format).	5-26
.AP	Assemble command (mnemonic conversion)	Rewrites a selected address of the IE-17K-ET program area in mnemonics. This is useful when using it with the UP command.	5-28
.DP	Program memory dump	Displays on the screen a maximum of a 3FH address size of the selected address range of the IE-17K-ET program area (1-4-3-4-4 format).	5-32
.UP	Disassemble command (mnemonic dump)	Displays up to 10 steps on the screen of a selected address of the IE-17K-ET program area (mnemonic format).	5-34
.FP	Program memory search	Searches the contents of the program memory in 1-4-3-4-4 format.	5-37
.SP0	Program memory save (channel 0)	Uses channel 0 of the RS-232-C to save the program in the IE-17K-ET. This is useful for saving patched programs.	5-38
.SP1	Program memory save (channel 1)	Uses channel 1 of the RS-232-C to save the program in the IE-17K-ET. This is useful for saving patched programs.	5-38

(to be continued)

(cont'd)

Command Name	Function	Summary	Page
.XS0	PROM data output (channel 0)	Uses channel 0 of the RS-232-C to change the program in the IE-17K-ET to a PROM file and then output it.	5-39
.XS1	PROM data output (channel 1)	Uses channel 1 of the RS-232-C to change the program in the IE-17K-ET to a PROM file and then output it.	5-39
.Q	Restart	If this command is executed when the prompt is >@@@, it is possible to restart the downloaded program.	5-40

B.2 DATA MEMORY CONTROL COMMANDS

Command Name	Function	Summary	Page
.ID	Data memory initialization	Initializes a selected range of the data memory with specified data.	5-43
.CD	Data memory change	Rewrites a selected address of the data memory one data at a time.	5-44
.DD	Data memory dump	Dumps the data memory of a selected range.	5-46
.D	Complete data memory dump	Dumps all of the data memory.	5-48

B.3 PERIPHERAL CIRCUIT CONTROL COMMANDS

Command Name	Function	Summary	Page
.GD	Display the peripheral register contents	Displays on the screen the contents of a peripheral register.	5-50
.GE	Read the peripheral register contents	Assigns the contents of the peripheral register in the Q-register. This is used when creating a user macro with the primitive command.	5-51
.PD	Write to the peripheral register	Assigns a numeric value to the peripheral register.	5-53
.PU	Indirect write to the peripheral register	Assigns the contents of the Q-register to the peripheral register. This is used when creating a user macro with the primitive command.	5-54

B.4 EMULATION COMMANDS

Command Name	Function	Summary	Page
.R	Reset	Resets the SE board. The program execution address becomes OH, and the data memory and register file are set to the same reset state as the target device.	5-56, 6-1
.RN	Program execution	Executes the specified program from the start address. The break and trace conditions are not changed.	5-57, 6-1
.BG	Program execution (reset some break and trace conditions)	Executes the specified program from the start address. (The break and trace conditions are partially reset.)	5-58
.BK	Break	Stops program execution.	5-59, 6-1
.CA	Program start address change	Changes the start address of the program.	5-60
.S	Step operation	The program is executed one instruction at a time by using the space key, for the number of times specified from the numeric input from the keyboard.	5-61, 6-18
.DS	Display	This command is for products that have an LCD controller. LCD display is possible during a break.	5-63

B.5 BREAK/TRACE CONDITION CONTROL COMMANDS

Command Name	Function	Summary	Page
.CC	Break/trace condition change	Sets or changes the break/trace condition.	5-65, 6-4
.CT	Trace ON/OFF condition change	Sets the starts and end of a trace, and performs the trace-one-shot setting.	5-81
.DC	Break/trace condition dump	Dumps the break and trace conditions onto the screen.	5-86
.DT	Trace table dump	Dumps the trace results.	5-88
.SC0	Break/trace condition save (channel 0)	Outputs the break/trace condition set by .CC level 1 to channel 0 of the RS-232-C.	5-92
.SC1	Break/trace condition save (channel 1)	Outputs the break/trace condition set by .CC level 1 to channel 1 of the RS-232-C.	5-92
.LC0	Break/trace condition load (channel 0)	Downloads the break/trace condition from channel 0 of the RS-232-C.	5-93
.LC1	Break/trace condition load (channel 1)	Downloads the break/trace condition from channel 1 of the RS-232-C.	5-93
.VC0	Break/trace condition verify (channel 0)	Used to verify the break/trace condition using channel 0 of the RS-232-C.	5-94
.VC1	Break/trace condition verify (channel 1)	Used to verify the break/trace condition using channel 1 of the RS-232-C.	5-94

B.6 COVERAGE DISPLAY COMMAND

Command Name	Function	Summary	Page
.DM	Coverage memory dump	Displays the passage history of the program memory, and the write history of the data memory.	5-96

B.7 HELP COMMAND

Command Name	Function	Summary	Page
.H	Support commands display	Displays a list of commands.	5-99