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.

RENESAS

User's Manual

LK17K

Linker

Document No. U12518EJ2V0UM00 (2nd edition)
(O.D.No. EEU-1540)
Date Published June 1997 J

© NEC Corporation 1995
Printed in Japan

SUMMARY OF CONTENTS

emlc-17K **and** ***SIMPLEHOST*** **are trademarks of NEC Corporation.**

MS-DOS and Windows are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

PC DOS and PC/AT are trademarks 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.

M7A 96.10

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 800-366-9782
Fax: 800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.1.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 01-504-2787
Fax: 01-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
United Square, Singapore 1130
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-719-2377
Fax: 02-719-5951

NEC do Brasil S.A.
Sao Paulo-SP, Brasil
Tel: 011-889-1680
Fax: 011-889-1689

Major Changes

Page	Description
Preface	Some related documents have been added.
P. 20	**Figure 2-13** has been modified.
P. 21	**Table 2-2** has been modified.
P. 31	**Section 2.1** has been modified.
P. 33	**Section 2.2** has been added.
P. 39	**Section 2.5** has been modified.
P. 93	Error message A406 has been added.

The mark ⋆ shows major revised points.

[MEMO]

PREFACE

Outline of product

LK17K is the linker of the Relocatable Assembler Package. It is used to link files created by RA17K and *emlC-17K*™.

Intended readers

This manual is aimed at those engineers working with the 17K series 4-bit single-chip micro controller, who are responsible for designing and developing related applications by using RA17K or *emlC-17K*.

Organization

This manual is organized as follows:

Language
Operation
Error messages

Prerequisites

Readers of this manual are assumed to be familiar with RA17K or *emlC-17K*.

Legend

This manual uses the following symbols and conventions:

... : Indicates that the preceding option can be repeated.
[ ] : The item enclosed in brackets is optional.
< > : Character or characters to be specified as is, usually a title enclosed in <>.
xxx : Indicates any character string.
Δ : Indicates a space.
Number representation systems : Binary : xxxxB
Decimal : xx or xxD
Hexadecimal : xxH

File naming rule

[drive-name:][\directory-name\...][filename[.extension]

A file name may include a drive name and directory name(s). A path name includes a drive name and directory name(s) only.

The software package may contain a README.DOC file. The README.DOC file provides information that was not available when the manual was printed. Always read the contents of the README.DOC file before attempting to use the product.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PART I

LANGUAGE

[MEMO]

CHAPTER 1 OUTLINE

1.1 FUNCTIONAL OUTLINE

LK17K takes, as its input, an object module file (.REL) from the relocatable assembler (RA17K) or compiler (*emIC-17K*). It produces the following files as its output:

- Link object module file (.LNK)
- Load module file (.ICE/.PRO)
- Differential file (.DIF)
- Link map file (.LMP)

The link object module file (.LNK) produced by LK17K cannot be reused as an input file for LK17K.

1.2 17K SERIES ARCHITECTURE

1.2.1 Program Memory

The program memory area is managed in units of segments, which consist of 8K steps. Names (SEG0 to SEG7) are assigned to segments. The 17K series allows up to eight segments to be mounted, but different devices mount a different number of segments. LK17K obtains the number of mounted segments from a device file.

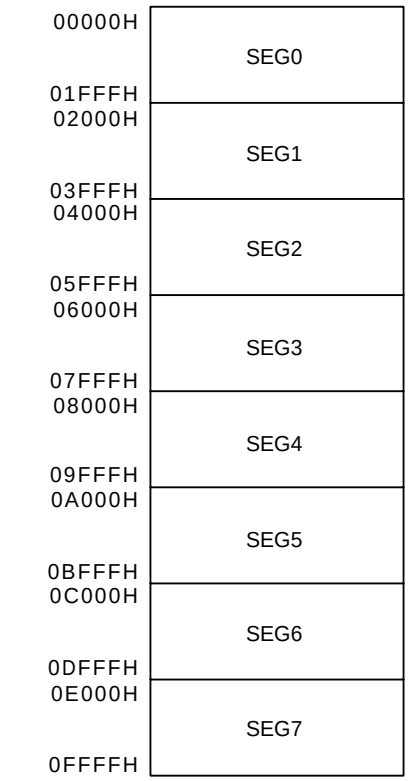
The program memory area supported by the 17K series is defined as follows.

Table 1-1. Program Memory Area Supported by the 17K Series (Maximum Configuration)

Memory area	Start address	Segment size
SEG 0 (segment 0)	00000H	2000H
SEG 1 (segment 1)	02000H	(8K steps)
SEG 2 (segment 2)	04000H	
SEG 3 (segment 3)	06000H	
SEG 4 (segment 4)	08000H	
SEG 5 (segment 5)	0A000H	
SEG 6 (segment 6)	0C000H	
SEG 7 (segment 7)	0E000H	

LK17K performs linkage (relocation) in units of sections. A section is an area defined by the RA17K CSEG dummy instruction. A section is allocated to the memory area mounted in the device. A section cannot be allocated to an address that is not mounted in the device.

Figure 1-1. Memory Area (Maximum Configuration)



PROGRAM space (64K steps = 128K bytes)

1.2.2 Segment Configuration

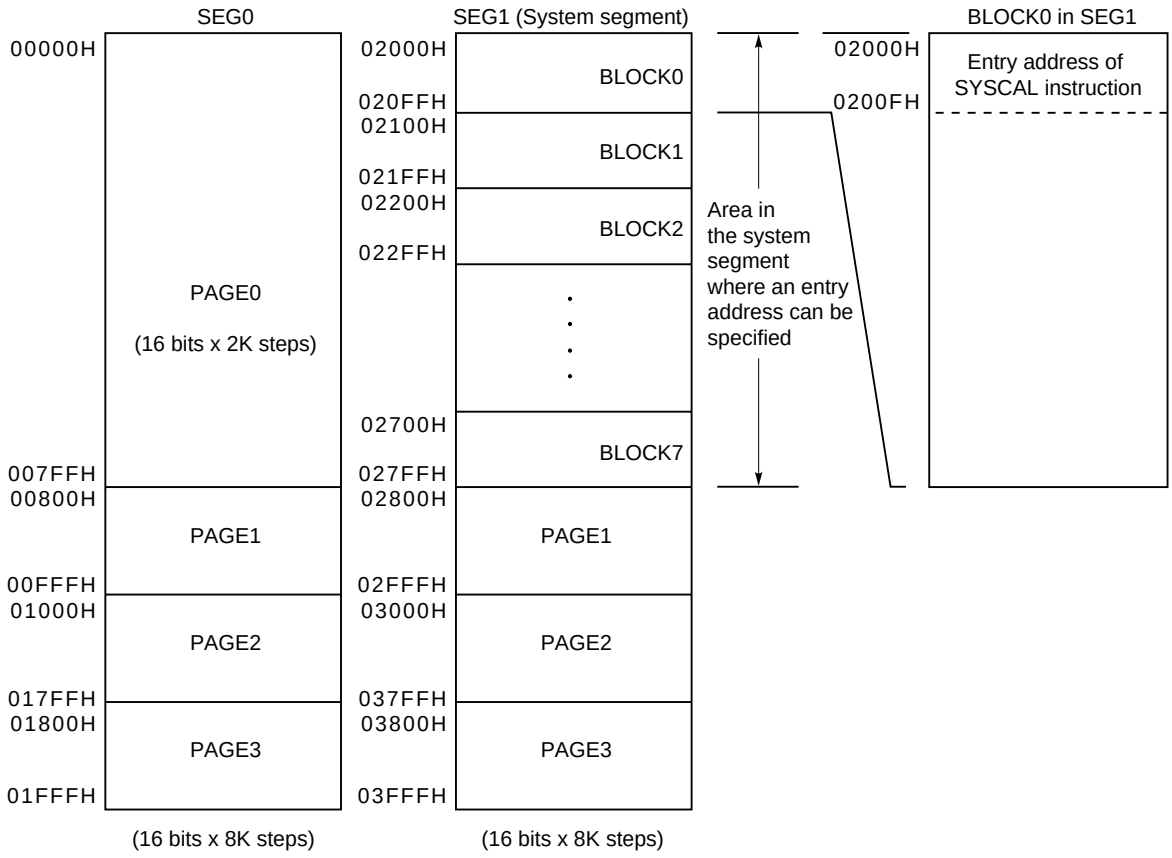
Each segment consists of pages (Pages), consisting of 2K steps. In those devices having more than one segment, the last segment can be handled as a segment to which control is branched using the SYSCAL instruction. This last segment is called a system segment. Page 0 of the system segment is further divided into blocks (BLOCK0 to BLOCK7) in units of 256 steps.

The 17K series supports both direct and indirect branch instructions. Direct branch instructions (BR addr and CALL addr) cannot cause a branch beyond the segment boundaries. This is because the maximum bit length of the program counter is 13 bits (8K steps). A segment is specified by the value in the segment register (SGR), provided separately from the program counter.

To cause a branch beyond the segment boundaries, use an indirect branch instruction (BR @AR or CALL @AR) with a system register, the address register (AR), or the SYSCAL instruction. One SYSCAL instruction can cause a branch to a system segment beyond the segment boundaries. The first 16 words of each block of the system segment are used for the entry address of the SYSCAL instruction.

Note that the CALL addr instruction can only cause a branch within Page0 because only a 11-bit address can be specified in an operand of the CALL addr instruction.

Figure 1-2. Example of 17K Segmented Address Space (16K Steps = 32K Bytes)



1.2.3 EPA Area

While debugging a program, the program size may temporarily exceed the ROM capacity of the product. It is particularly inconvenient if those portions of a program that exceed the ROM capacity cannot be debugged. With the IE-17K in-circuit emulator for the 17K series, because the chip's program counter is used during debugging, the bit length of the program counter will become insufficient if the program size exceeds the ROM capacity of the product. As a result, those portions of the program that exceed the capacity cannot be the controlled.

To overcome this problem, the EPA (Extra Program Address) bit is now supported by the IE-17K, allowing it to control a program of a size up to double that of the ROM capacity of the chip.

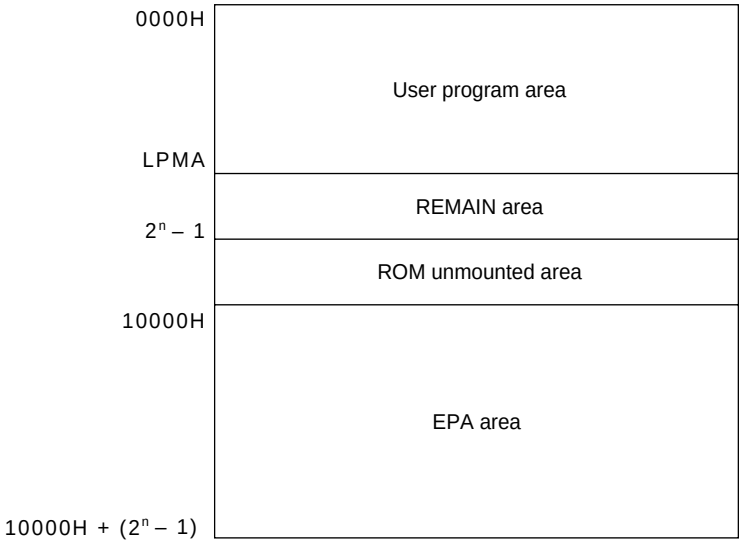
One bit, the EPA bit, is added to the address sent from the program counter of this chip to add one bit to the program counter. Thus, the size of the address space that can be controlled by this chip is doubled. This allows the IE-17K to debug a program even if the program size exceeds the ROM capacity.

Because the EPA bit is added at a position higher than that of the most significant bit of the product's ROM address, the ROM capacity which can be emulated is doubled. The address space for which the EPA bit is set to 1 is called the EPA area. It is used as a patch area (Patch Area).

If the last ROM address (LPMA) of the product does not exceed $2^n - 1$, the address space in excess of LPMA and up to $2^n - 1$ constitutes an emulation area which can be handled without the EPA bit. This area is called the REMAIN area. It is used as a patch area in the same way as the EPA area.

If the program exceeds the effective address of ROM, LK17K causes an error. This error differs from an ordinary error in that a normal object is created in spite of the invalidity. When ordering ROM code, keep reducing the program size until this error no longer occurs.

Figure 1-3. Address Space in a 17K Series Segment



CHAPTER 2 FUNCTIONS

This chapter explains the functions of LK17K.

2.1 DIRECTIVES

2.1.1 Outline

LK17K performs linkage (relocation) in minimum units of sections. A section is an area defined by the RA17K CSEG dummy instruction.

Directives are a group of instructions which are used to instruct the LK17K to perform various operations at link time, such as allocation of input files, available memory areas, and sections. A directive is used as follows:

<1> The user describes directives in the directive file, described below.
<2> Specify the name of the directive file in the linker option (-DIR) when starting LK17K.

LK17K reads the directive file and performs linkage while interpreting the directives described in it.

Directives include a section allocation directive, used to specify the address to which a section is to be allocated.

2.1.2 Directive File

A directive file is a text file containing the directives used to instruct LK17K to allocate sections. For an explanation of the directive description format, see **Section 2.1.5**.

The following are reserved words for a directive file.

```
MERGE
AT DSBR DSYS SBR SSYS BOOT VECTn DVECTn CROM
SEQUENT COMPLETE
SEG0 SEG1 SEG2 SEG3 SEG4 SEG5 SEG6 SEG7
```

In a directive file, reserved words cannot be used to imply other meanings (such as section names).

Reserved words may be specified in either upper or lower case. Upper and lower case letters cannot, however, be mixed.

Only one directive file can be specified for LK17K. An error occurs if two or more directive files are specified.

More than one directive can be described in a single directive file.

[Example of directive file description]

TEST.DR

```
MERGE SEC1:BOOT
MERGE SEC2:SBR = SEG0
MERGE SEC3:AT (0100H)
```

2.1.3 Numeric Values

To enter numeric constants in a directive (to specify a relocation attribute definition with an absolute address, for example), code them using decimal or hexadecimal rotation.

2.1.4 Comment Statements

When a ";" or "#" appears in a directive file, the subsequent characters up to a line feed (LF) or end-of-file (EOF) code are handled as a comment.

2.1.5 Section Allocation Directive (Merge Directive)

The merge directive is used to allocate a specified section to a specific address on memory.

[Syntax]

MERGE Δ <section-name> [Δ] : [Δ] [<relocation-attribute-definition>][Δ<merge-type-definition>] [[Δ] = [Δ] <memory-area-specification>]

(1) <section-name>

<section-name> is a section name contained in an object file. Only a section name contained in an object file can be specified as <section-name>.

<section-name> is case-sensitive and must, therefore, be specified exactly as it appears in the source file.

(2) <relocation-attribute-definition>

The following can be specified as <relocation-attribute-definition>.

[Relocation attribute definition]

AT [Δ] ([Δ] <start-address> [Δ])	; Absolute address specification
BOOT	; Startup routine specification
VECTn	; Indirect interrupt processing routine specification (creates a BR instruction at an interrupt vector address)
DVECTn	; Direct interrupt processing routine specification (does not create a BR instruction at an interrupt vector address)
CROM	; Specification of data in a non-program area (such as DTS CROM)
DSYS	; Direct system subroutine specification (does not create a BR instruction in the entry address of the SYSCAL instruction)

SSYS	; System subroutine specification (creates a BR instruction in the entry address of the SYSCAL instruction)
DSBR	; Direct subroutine specification (does not create a BR instruction in the entry address of the CALL instruction)
SBR	; Subroutine specification (creates a BR instruction in the entry address of the CALL instruction)

Caution The above reserved words must be coded in either upper or lower case only. An error occurs if upper and lower case letters are mixed.

<start-address> must be specified with a numeric constant. A symbol cannot be used.

n in VECTn/DVECTn is a vector address for each interrupt source ($1 \leq n \leq$ Maximum vector address). The maximum vector address differs depending on the device. An address must be specified in hexadecimal; however, an “H” indicating hexadecimal rotation must not be appended.

The following explains the allocation method for each relocatable attribute.

Figure 2-1. Relocatable Attribute (AT) and Allocation Method

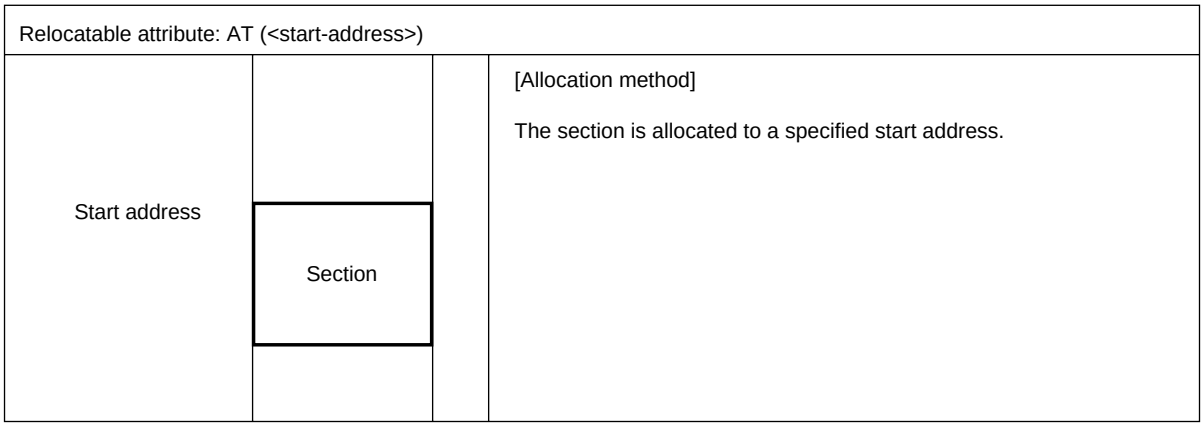


Figure 2-2. Relocatable Attribute (BOOT) and Allocation Method

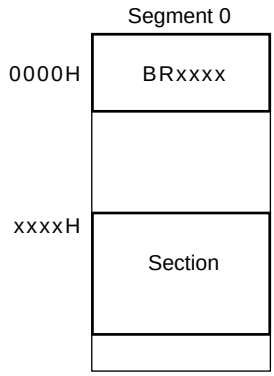
Relocatable attribute: BOOT	
	[Allocation method] A branch instruction 1 instruction, causing a branch to the BOOT section block, is created at address 0 in PAGE0 of segment 0. The BOOT section block is allocated to any address in segment 0. The BOOT section block is assigned a higher allocation priority than sections having other relocation attributes. PAGE: 2K steps Segment: 8K steps

Figure 2-3. Relocatable Attribute (CROM) and Allocation Method

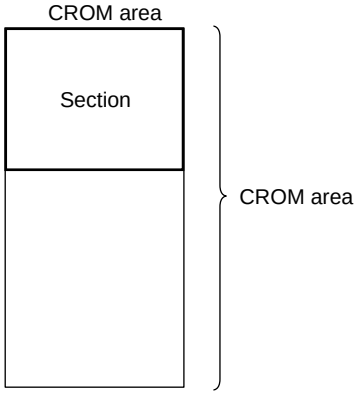
Relocatable attribute: CROM	
	[Allocation method] The CROM section blocks are allocated sequentially starting from the top address defined in the CROM area definition of the device file.

Figure 2-4. Relocatable Attribute (VECTn) and Allocation Method

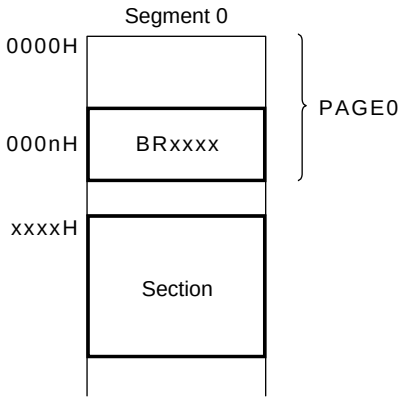
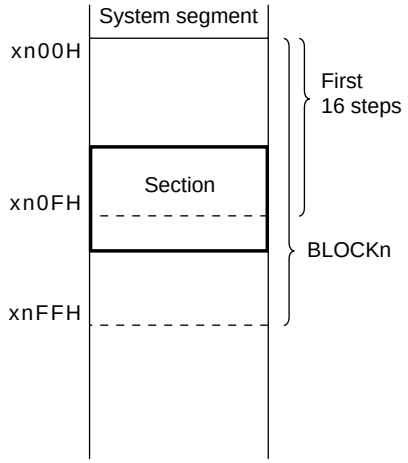
Relocatable attribute: VECTn	
	[Allocation method] A branch instruction 1 instruction, causing a branch to the VECTn section block, is created at vector address n of PAGE0 in segment 0. The VECTn section block is allocated to any address in segment 0. The VECTn section block having the largest value of n is allocated directly from vector address n. Specify the -IND option to allocate it indirectly.

Figure 2-5. Relocatable Attribute (DSYS) and Allocation Method

Relocatable attribute: DSYS



[Allocation method]

Section blocks are allocated sequentially so that the top of each block fits in the first 16 steps of each block of PAGE0 in the system segment.

However, they are allocated so that the top address of the section is aligned with the most significant address of the first 16 words.

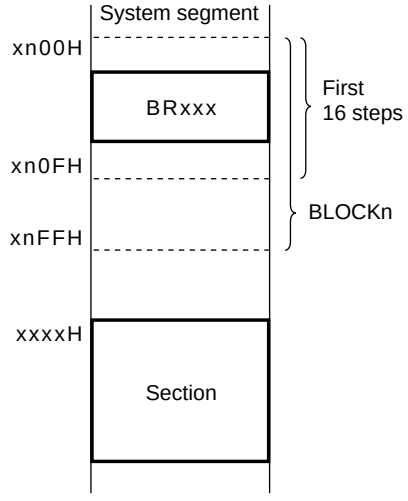
Because each entry in the system segment consists of the first 16 steps of each block in PAGE0, the allocation addresses may become non-consecutive.

A section having this relocatable attribute is assigned a higher allocation priority than a section having the SYS attribute.

BLOCK: 256 steps

Figure 2-6. Relocatable Attribute (SSYS) and Allocation Method

Relocatable attribute: SSYS



[Allocation method]

Branch instructions which cause branches to section blocks are created sequentially in the system call entries located in the first 16 steps of each block of PAGE0 in the system segment.

The SSYS section block is allocated to an area other than the first 16 steps of each block.

If, however, the block cannot be allocated within one block, it is allocated to any address other than an address in PAGE0.

Figure 2-7. Relocatable Attribute (DSBR) and Allocation Method

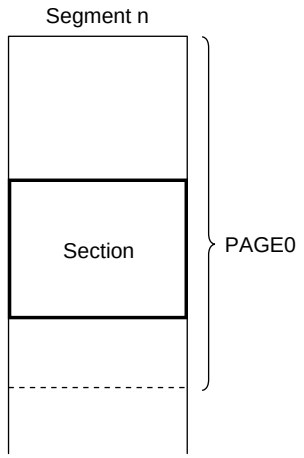
Relocatable attribute: DSBR	
	[Allocation method] Sections are allocated sequentially, starting from the top address of PAGE0 in the specified segment. If there are sections for which BOOT or VECTn is specified, these sections are allocated first and the sections are allocated sequentially starting from the subsequent address. If a section cannot be allocated to PAGE0, an error occurs and this section is not allocated. This relocation attribute has a higher priority than the other relocatable attributes except BOOT and VECTn.

Figure 2-8. Relocatable Attribute (SBR) and Allocation Method

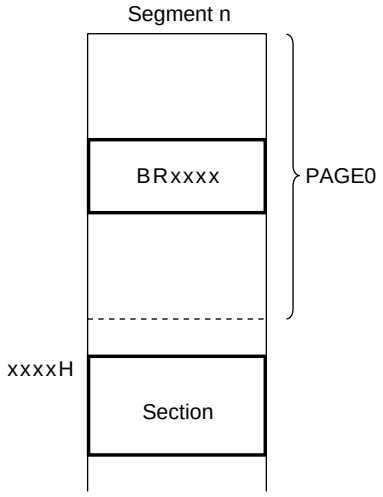
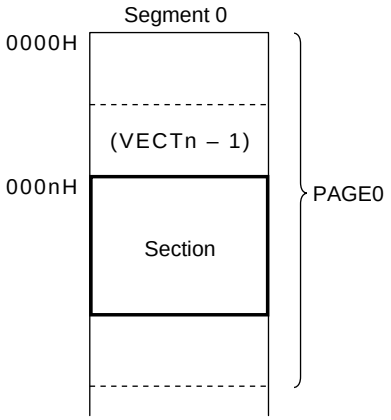
Relocatable attribute: SBR	
	[Allocation method] Branch instruction 1 instructions are created and allocated sequentially, starting from the top address of PAGE0 in the specified segment. The section blocks are allocated to any addresses in the specified segment. If there are any sections for which BOOT, VECTn, or DSBR is specified, these sections are allocated first, after which the instructions are allocated sequentially starting from the subsequent address. The difference from DSBR is in the allocation priority.

Figure 2-9. Relocatable Attribute (DVECTn) and Allocation Method

Relocatable attribute: DVECTn



[Allocation method]

DVECTn section block is allocated to vector address n in PAGE0 of segment 0.

This relocatable attribute has a higher priority than VECTn.

If the section block cannot be allocated to vector address n, an error occurs and the section is not allocated.

[Cautions regarding relocatable attribute definitions]

<1> When <relocatable-attribute-definition> is omitted, the section is allocated to any address in a memory area.

<2> If a section is too large for a memory area, when allocated starting from the specified <start-address>, the section will be allocated to the EPA area. If it is also too large for the EPA area, an error occurs.

<3> An error occurs when more than one <relocatable-attribute-definition> is specified in a single merge directive. If this occurs, an error message is output and the other directives are checked for errors. After the directives have been interpreted, LK17K stops execution.

<4> When a section is allocated according to <relocation-attribute-definition>, and part of the section is allocated to the EPA area, a new section is created for the part allocated to the EPA area. The name of the created section will be as follows:

@ <section-name> <section-name>: Name of the section to be allocated to the EPA area (name of the original section for which <relocation-attribute-definition> is specified)

<5> When a branch instruction 1 instruction is created in response to <relocatable-attribute-definition>, a new section is created for the branch instruction. The name of the section produced will be as follows:

? <section-name> <section-name> : Name of the section for which a branch instruction 1 instruction is to be created (name of the original section for which <relocatable-attribute-definition> is specified)

(3) <merge-type-definition>

<merge-type-definition> specifies how the specified section is to be linked when an input section(s) having the same name already exists.

The types specifiable as <merge-type-definition> and their meanings are given below. For details of merge types, see **Section 2.2**.

[Merge type definition]

SEQUENT	Sections having the same name are linked sequentially in the order in which they were input to LK17K (default).
COMPLETE	An error occurs when a section(s) with the same name as the section already exists.

Reserved words SEQUENT and COMPLETE must be written in either upper or lower case. An error occurs if upper and lower case letters are mixed.

[Cautions regarding the merge type definition]

<1> When <merge-type-definition> is omitted, SEQUENT is assumed.

<2> An error occurs when more than one <merge-type-definition> is specified in a single merge directive. If this should occur. Once error message is output and the other directives are checked for errors. Once the directives have been interpreted, LK17K stops.

(4) <memory-area-specification>

Specifies a program memory area to which the section is to be allocated.
The following eight options can be specified as <memory-area-specification>.

SEG0 (segment 0), SEG1 (segment 1), SEG2 (segment 2), SEG3 (segment 3), SEG4 (segment 4), SEG5 (segment 5), SEG6 (segment 6), and SEG7 (segment 7)

[Cautions regarding memory area specification]

<1> Those sections containing no <memory-area-specification> are allocated sequentially, starting from a free address.

<2> An error occurs in the following cases. In this case, an error message is output and the other directives are checked for errors. Once the directives have been interpreted, LK17K stops.

- More than one <memory-area-specification> is specified.
- Options other than SEG0, SEG1, SEG2, SEG3, SEG4, SEG5, SEG6, and SEG7 are specified as <memory-area-specification>.

[General cautions regarding the merge directive]

<1> An input section that is not specified in the merge directive is allocated to any address at link time.

<2> An error occurs in the following cases. In this case, an error message is output and the other directives are checked for errors. Once the directives have been interpreted, LK17K stops.

- The section specified as <section-name> does not exist.
- A merge directive is specified more than once for a single section.

[Example of directive file description]

TEST.DR

```
MERGE SEC1:BOOT
MERGE SEC2:SBR = SEG0
MERGE SEC3:AT (0100H)
```

2.2 LINKING (MERGING) INPUT SECTIONS

This section explains how sections are linked.

2.2.1 Section Merge Types

Each section has a merge type. A section merge type specifies how a section is to be linked. There are two section merge types;

- SEQUENT (default)
- COMPLETE

2.2.2 Determining a Merge Types

A section merge type is determined as follows.

(1) If a merge type is explicitly specified in a directive at link time, a section will have the merge type specified in the directive.
(2) If a merge type is not specified in a directive at link time, a section will have the merge type SEQUENT.

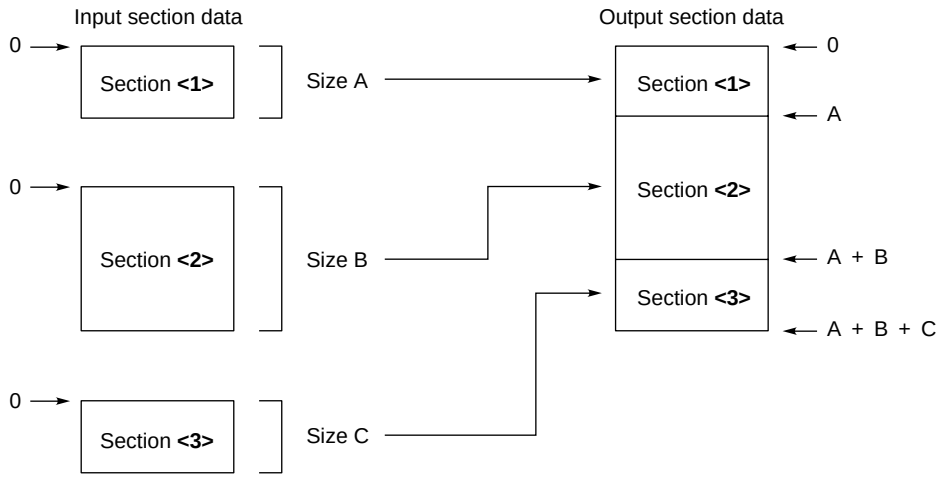
2.2.3 Merge Method for Each Merge Type

The following explains the section merge method for each merge type.

(1) For merge type SEQUENT

Sections are merged sequentially in the order in which they appear in the linker, leaving no spaces.

Figure 2-10. Merge Method for Merge Type SEQUENT



(2) For merge type COMPLETE

An error occurs when input sections having the same name exist.

2.3 SECTION RELOCATION ATTRIBUTES

After merging sections, LK17K determines section allocation addresses.
This section explains how section allocation addresses are determined.

2.3.1 Determining Relocation Attributes

Section relocation attributes are determined as follows.

(1) When a relocation attribute is specified in a directive at link time, the specified relocation attribute is assumed.
(2) When a relocation attribute is not specified at link time, the relocation attribute specified in the assembler source program is assumed.
(3) If no relocation attribute is specified in the assembler source program and no relocation attribute is specified at link time, the section is allocated to any address in memory.

With some relocation attributes, LK17K automatically creates branch instructions (BR addr). The following explains which relocation attributes cause branch instructions to be created and where the section blocks are allocated.

Table 2-1. Whether Branch Instructions to Branch to Sections Are Created Due to Relocation Attributes

Relocation attribute	Branch instruction (BR addr)	Where section blocks are allocated
AT	Not created	Specified address
BOOT	Created (created at address 0 in SEG0)	SEG0
VECTn (direct)Note 1	Not created	Address n in SEG0
VECTn (indirect)	Created (created at address n in SEG0)	SEG0
DVECTnNote 1	Not created	Address n in SEG0
CROM	Not created	SEG0 to SEG7 (device-specific)
DSYS	Not created	Within the first 16 steps of each BLOCK of PAGE0 in the system segment
SSYS	Created (created within the first 16 steps of each BLOCK of PAGE0 in the system segment)	SEG0 to SEG7 (system segment) (device-specific)
DSBR	Not created	PAGE0 of SEG0 to SEG7
SBR	Created (created in PAGE0 of SEG0 to SEG7)	Same segment as the branch instruction
TABLENote 2	Not created	SEG0-SEG7
None specified	Not created	SEG0-SEG7

Notes 1. When linked with -IND specified, the section is forced to be allocated indirectly.
2. The TABLE attribute is automatically assigned to the table block defined as TABLE by the CSEG dummy instruction of the assembler. The attribute cannot be specified in a merge directive.

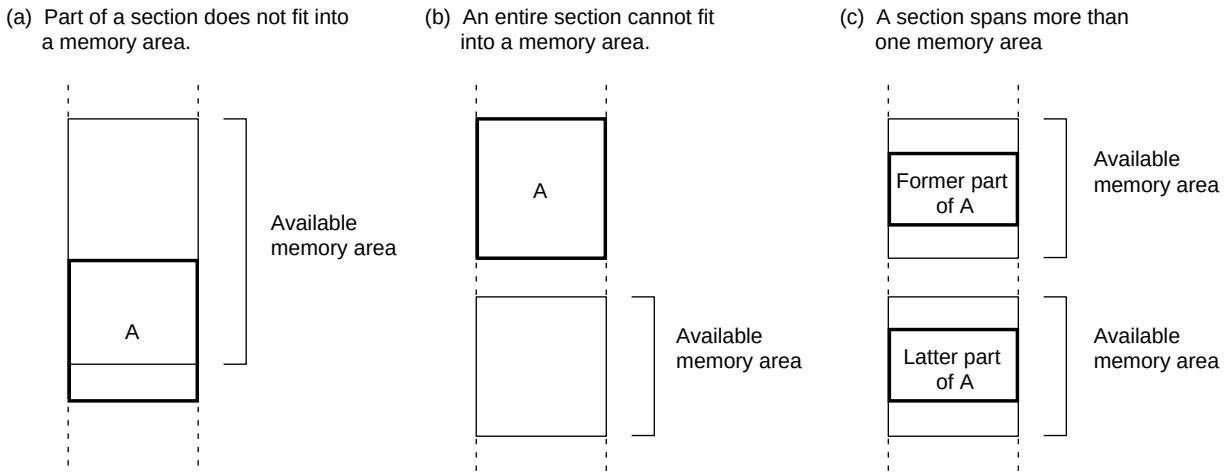
2.3.2 Determining Section Allocation Addresses

Section allocation addresses are determined as follows.

(1) Each section is allocated such that that section will fit into one memory area. Thus, a section is not allocated, as described below. If such an allocation is specified, an error message is output and the linker stops.

- Part or all of a section can not fit into a memory area
- A section is allocated to non-consecutive addresses (spanning more than one available memory area).

Figure 2-11. Examples of Invalid Allocation

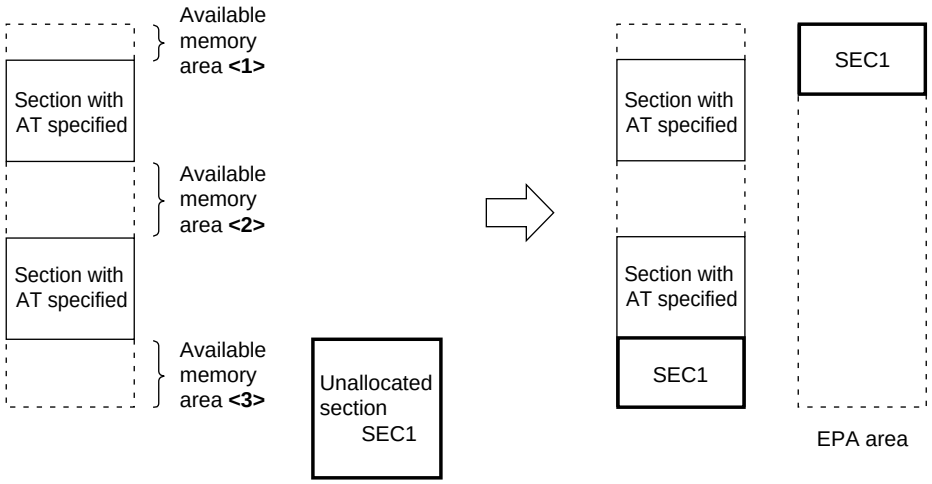


(2) When a memory area is specified for a section in a merge directive, the section is allocated to the memory area.

(3) All sections have relocation attributes. LK17K determines allocation addresses according to the relocation attributes. For an explanation of the rules governing how allocation addresses are determined for each relocation attribute, see **Section 2.3.3**.

(4) When a section is to be allocated to the last available memory area in the memory space, any part of the section that exceeds the available memory area is allocated to the EPA area.

Figure 2-12. Allocating a Section to the EPA Area



2.3.3 Allocating Sections

The following explains the procedure for allocating sections.

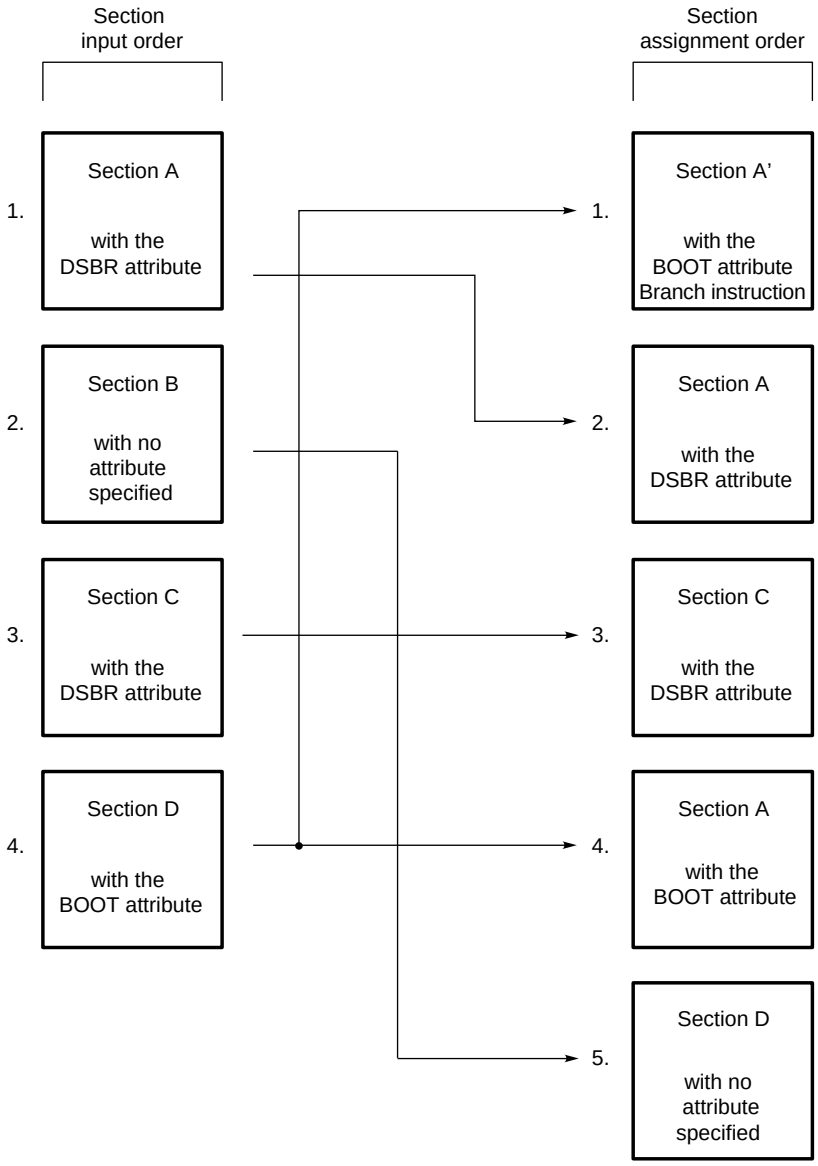
(1) The allocation address for a section having a relocation attribute with a high allocation priority is determined prior to that for a section having a relocation attribute with a lower priority. For details of the relocation priority, see **Table 2-2**.

(2) Where more than one section has the same relocation attribute, the addresses of those sections are determined in the order in which the sections are input.

(3) If a section can be assigned to more than one memory area, it is assigned to the area having the lowest address.

*

Figure 2-13. Example of Section Allocation



* **Table 2-2. Relocation Priority**

Priority	Section or instruction to be relocated
1	Section having the AT attribute Branch instruction created by LK17K, in response to the specification of the BOOT attribute
2	Section having the VECTn attribute, to which an entity will be directly allocated Section having the DVECTn attribute
3	Branch instruction created by LK17K, in response to the specification of the VECTn attribute
4	Branch instruction created by LK17K, in response to the specification of the SSYS attribute
5	Section having the CROM attribute Section having the DAYS attribute
6	Section (entity) having the SSYS attribute
7	Section having the TABLE attribute (This attribute cannot be defined by LK17K, only by the assembler.) Branch instruction created by LK17K, in response to the specification of the SBR attribute
8	Section having the DSBR attribute
9	Section (entity) having the BOOT attribute Section (entity) having the VECTn attribute
10	Section (entity) having the SBR attribute
11	Section with no relocation attribute, but with a memory area specified Section having no relocation attribute nor memory area specified

Remark Where more than one section has the same priority, the order in which relocation is performed corresponds to the order in which the sections are input to LK17K.

2.4 DETERMINING AND OUTPUTTING SYMBOL VALUES

LK17K performs the following processing for symbols.

(1) Checks the reference relationship between external definition symbols and external reference symbols
(2) Determines, then outputs, symbol values

2.4.1 Checking the Reference Relationship Between External Definition Symbols and External Reference Symbols

LK17 checks the definition and reference relationship between external definition symbols (symbols declared as external definitions with the PUBLIC dummy instruction in the assembler source program, referred to as PUBLIC symbols in this manual) and external reference symbols (symbols declared as external references with the EXTRN dummy instruction in the assembler source program, referred to as EXTRN symbols).

The definition and reference relationship is checked as follows:

(1) Checking for duplicate definitions

LK17K ensures that no duplicate PUBLIC symbols are defined in the input object module files. If duplicate symbols are defined, an error message is output and link processing stops.

(2) Checking for undefined symbols

LK17K checks that, for each EXTRN symbol, there is a PUBLIC symbol having the same name among the input object module files. If, for a given EXTRN symbol, there is no PUBLIC symbol with the same name, an error message is output. In addition, if -JUNK is specified, an object file is created using the object code output by the assembler for the referenced instruction. Relocation remains unsolved.

2.5 ERROR MESSAGES OUTPUT UPON A PROGRAM MEMORY OVERFLOW

17K can debug those portions of a program that exceed the ROM capacity (see **Section 1.2.3**.). To support this, LK17K creates the object code for a program using an address area twice as large as the ROM capacity. However, because a program cannot run on an actual device if its size exceeds the ROM capacity, the following error messages will be output. When ordering ROM code, keep reducing the program size until these errors are no longer output.

(1) If the program is in area <2>

At the point where area **<1>** is exceeded, an error (F306) occurs for that section. The object code, however, is created correctly.

(2) If the program is in area <4>

When the entire section is in area **<4>**, error (F317) occurs. If only part of the section is in area **<4>**, error (F318) occurs. In the latter case, however, the object code is created correctly.

(3) If the program is in area <3> or exceeds area <4>

Error (F206) occurs. No object code is created.

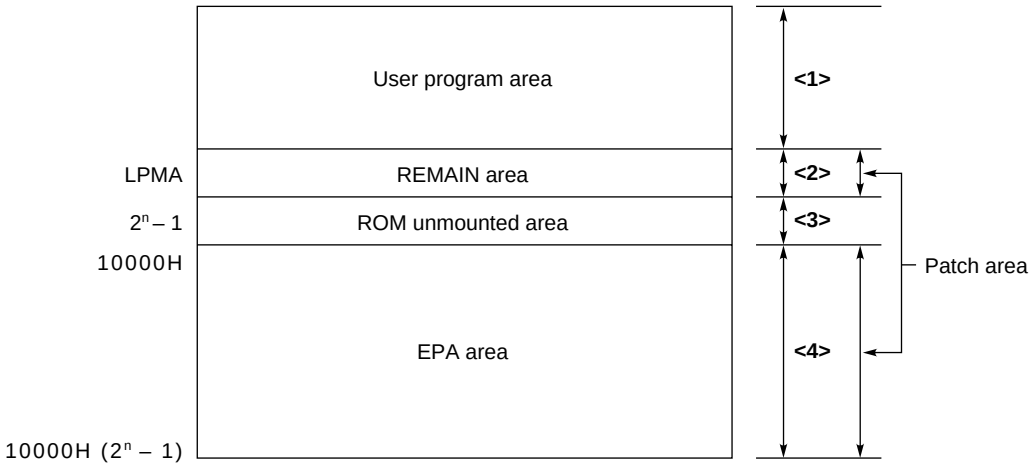
(4) If the jump destination address is in area <2> (REMAIN AREA)

Error (F309) occurs.

(5) If the jump destination address is in area <4> (EPA AREA)

Error (F310) occurs.

Figure 2-14. Address Space in a Segment for the 17K Series



Remark LPMA (Last Program Memory Area): Last address in the program memory mounted in a product

2.6 DIFFERENTIAL FILES FOR INCREMENTAL LOAD

2.6.1 Creating Differential Files

SIMPLEHOST, which is compatible with RA17K/*emIC-17K*, reassembles (recompiles) modified modules only. When linker option -INC is specified, LK17K outputs only the differences in the object code/debug information to a differential file. By loading this differential file to IE-17K, the time required for patch processing can be reduced.

(1) Cautions related to outputting a differential file

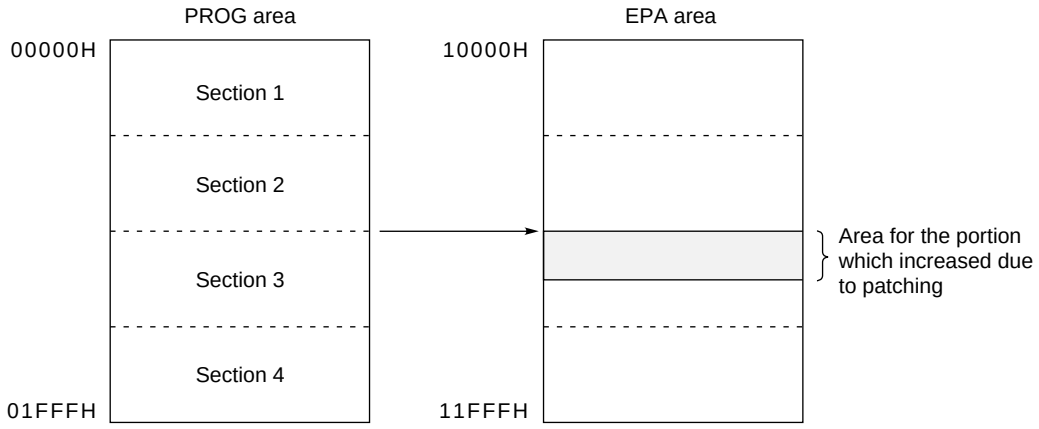
<1> -INC cannot be specified with -NOOB or -NOIC.

<2> When -INC is specified, those sections which existed at the last link time must not be deleted with the source modification. Otherwise, an error will occur.

(2) Example of outputting a differential file

The following briefly explains the contents of a differential file produced after the object code and size are changed by patching.

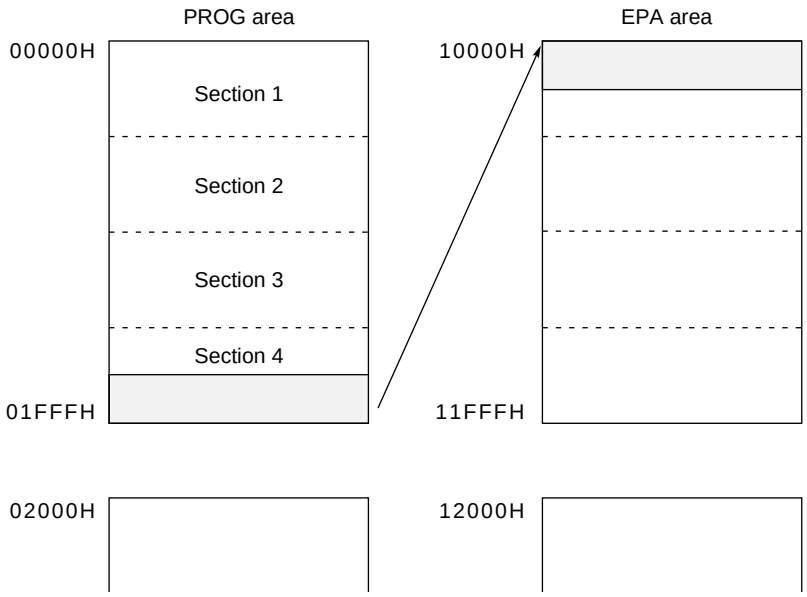
Example 1. Section 2 is patched, causing section 2 to increase in size.
All of that part of the object code/EPA bit map subsequent to the first difference detected in the object code/EPA bit map is output to a differential file.
If section 2 increases in size due to the patching of section 2, the code for the increased portion is allocated to the EPA area for section 3.



Example 2. Section 4 (allocated to the highest address in a segment) is patched, such that section 4 increases in size.

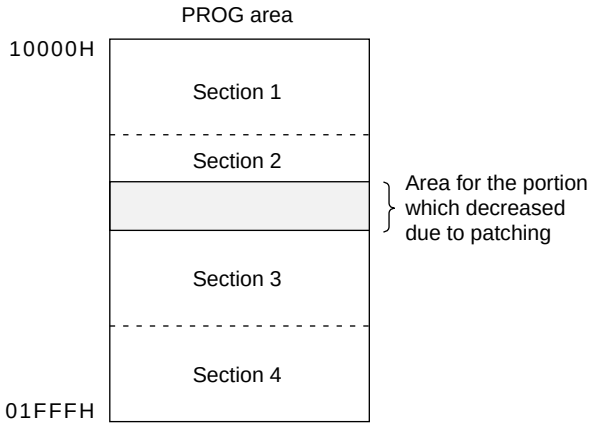
All of that part of the object code/EPA bit map subsequent to the first difference detected in the object code/EPA bit map is output to a differential file.

If section 4 increases in size due to the patching of section 4, the code for the increased portion is allocated, starting from the consecutive free areas in the program area. If it exceeds the last address of the program area, the remainder is allocated starting from the EPA area (10000H or above) of the segment. A branch from the last address (in segment units) of the program area to the beginning of the EPA area in the segment is controlled by adding the EPA bit to the instruction in the last address of the program area. Therefore, the EPA bit map for the instruction in the last address is set to “1”.

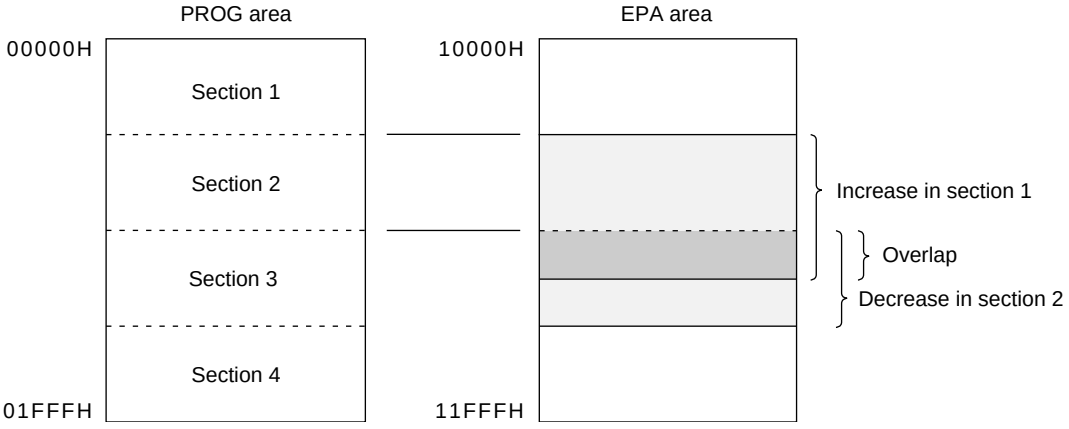


Remark ▭ indicates the area for the portion which increased due to patching.

Example 3. Section 2 is patched, thus section 2 decreases in size.
All of that part of the object code/EPA bit map subsequent to the first difference detected in the object code/EPA bit map is output to a differential file.



Example 4. Consecutive sections are patched, such that the patched portion of one section exceeds the size of the subsequent section (multiple sections are patched).
When consecutive sections 1 and 2 are patched and both sections increase in size, ensure that the patch portion of section 1 does not exceed the size of section 2. Otherwise, the two patch portions will overlap, making correct patching impossible. Should this occur, the linker will cause an error while attempting to create a differential file; no differential file is created.
When multiple sections are patched, a differential file will be correctly output provided their patch portions do not overlap.



PART II

OPERATION

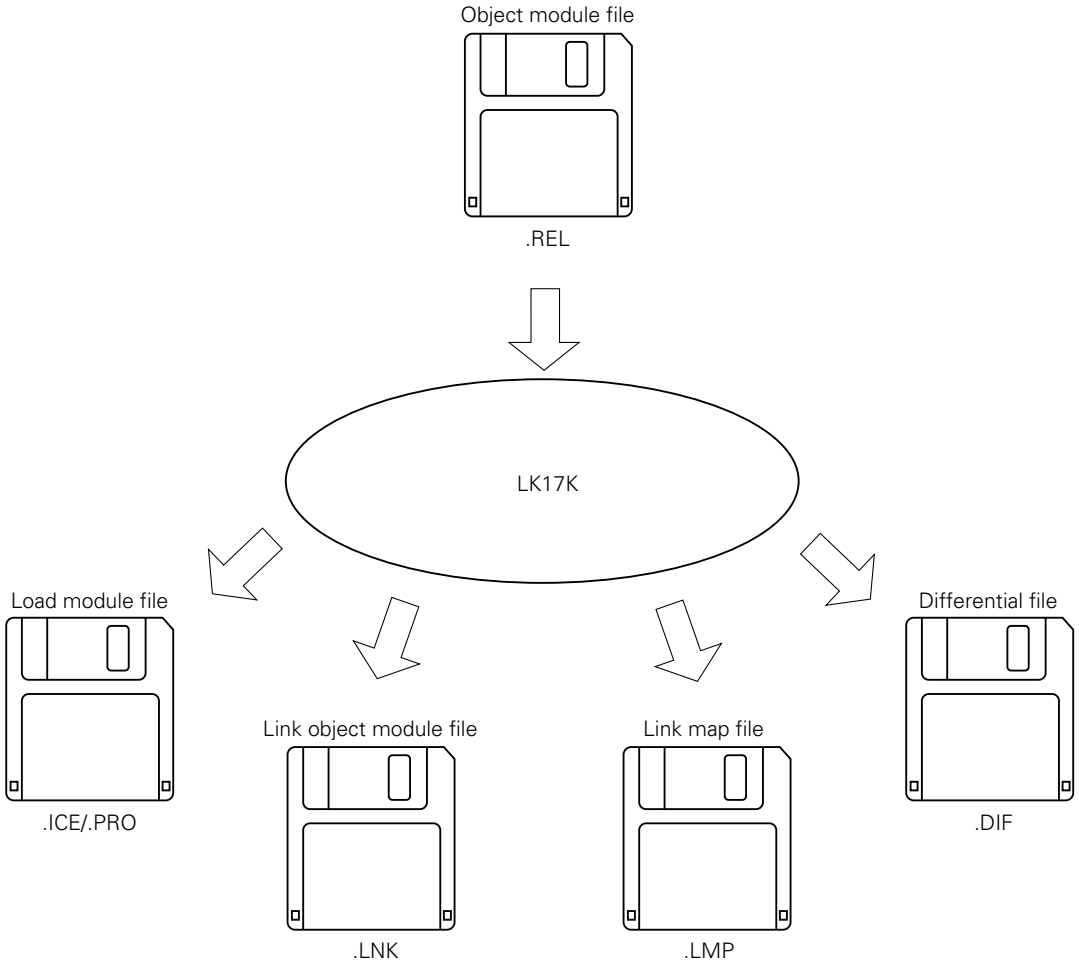
[MEMO]

CHAPTER 1 OUTLINE

LK17K inputs the object module file (.REL), created by the assembler (RA17K)/compiler (*emIC-17K*), and outputs the following files:

- Load module file (.ICE/.PRO)
- Link object module file (.LNK)
- Link map file (.LMP)
- Differential file (.DIF)

Figure 1-1. Outline of Processing



[MEMO]

CHAPTER 2 SYSTEM CONFIGURATION

* 2.1 SYSTEM ENVIRONMENT

LK17K operates in the environment described below.

2.1.1 Hardware Environment

(1) Host machine

<1> PC-9800 series

<2> PC/AT™

(2) OS

MS-DOS™ : Ver. 3.30 or later

PC DOS™ : Ver. 5.02 or later

Windows™ 3.1, Windows 95

(3) CPU

80386 or better

(4) Minimum memory size required for running the system

Conventional memory : 300 KB or more

Protect memory : 1.5 MB or more

(5) Floppy disk drive

2.1.2 Software Environment

(1) Command line environment

As the memory driver, at least himem.sys or an equivalent driver is required.
Operation in either of the following environments must be guaranteed.

- Environment in which only himem.sys is used
- Environment in which himem.sys and emm386.exe are used

(a) PC-9800 Series

Environment / OS	himem.sys.only	himem.sys and emm386.exe
MS-DOS 3.30D	○Note 1	xNote 1
MS-DOS 5.00A	○	x
MS-DOS 6.2	○	○Note 2, Note 3
Windows 95 (dos prompt only)	○	○Note 2

Notes 1. The drivers provided with Window 3.1 are assumed to be used because himem.sys and emm386.exe are not provided with MS-DOS 3.30D.
2. The /DPMI switch must be added to the end of emm386.exe.
3. If the DPMI server provided with DOS is installed, the operation of LK17K cannot be guaranteed.

Note that emm386.exe, provided with MS-DOS 5.00A or Windows 3.1, does not support the /DPMI function. Therefore, to allocate the UMB/EMS, use emm386.exe together with a third-party product. Or, use it in a Windows 3.1 DOS window instead of executing it from the command line.

(b) PC/AT or AT compatibles

Environment / OS	himem.sys.only	himem.sys and emm386.exe
MS-DOS 5.0	○	○
MS-DOS 6.3	○	○
MS-DOS 6.2	○	○
MS-DOS 7.0	○	○
Windows 95 (dos prompt only)	○	○

(2) DOS window environment

(a) Common to the PC-9800 Series, PC/AT, and AT compatibles

Environment / OS	himem.sys.only	himem.sys and emm386.exe
Windows 3.1	○Note	○Note
Windows 95	○	○

Note The windpmi.386 driver must be specified in the [386Enh] field of system.ini. The windpmi.386 driver can be installed using the installer provided with the RA17K assembler package.

* 2.2 FILE CONFIGURATION

To activate LK17K, the following file configuration is required.

(1) Linker

- LK17K.EXE: Main body of LK17K (32-bit application)

(2) Attached files for DOS-Extender

The following three files, which are Borland C attachments, are provided:

- 32RTM.EXE
- DPMI32VM.OVL
- WINDPMI.386

The above files can be redistributed.
The files are used in common for other 32-bit applications in the RA17K assembler package.

2.3 INPUT/OUTPUT FILES

2.3.1 Input/Output File List

The following table lists the files handled by LK17K.

Table 2-1. Input/Output File List

File/abbreviation	Input/output	Output destination	Explanation
Object module file (.REL)	Input	–	Binary image file of the object code produced by assembly
Device file (.DEV)	Input	–	File storing information specific to each product
Directive file (.DR)	Input	–	File describing link instructions for LK17K
Parameter file (.PLK)	Input	–	File describing parameters for LK17K
Link object module file (.LNK)	Output	File or output device	Binary image file of the object code resulting from linking
Load module file	Output	File or output device	.ICE : Load module file in Intel HEX (.ICE, .PRO) format, including debug information
			.PRO: Load module file in Intel HEX format, which does not include debug information
Temporary file (LKxxxxxx)	Input/output	File or output device	Intermediate file for linking
Log file (LK17K.LOG)	Output	File or output device	File storing the messages output to the display during link time. The file name is fixed to LK17K.LOG.
Differential file (.DIF)	Output	File or output device	File produced when the incremental link option -INC is specified. File storing the differences between the new and old .ICE files
Link map file (.LMP)	Output	File or output device	File storing the results (allocation information and symbol information) of linking.

2.3.2 Output Destinations

There are two different output destinations:

- File
- Output device

(1) File

[Format]

[<path-name>] <file-name>

or

[<drive-name> :] [\] [[<directory-name> \] ...] <primary-name> [. [<file-type>]]

[Explanation]

If the file is not in the current drive and directory, the file must be specified with <path-name> plus <file-name>.

If <path-name> is omitted, the file is assumed to exist in the current drive and directory.

(2) Output device

[Format]

<device-name>

[Explanation]

<1> The following output devices can be specified as an output destination.

Table 2-2. Output Devices

Device name	Output destination
CON	Console
PRN	Printer Cannot be specified as the output destination for a binary file.
NUL	Null device
AUX	Serial interface (RS-232C)

2.3.3 Specifying Input Files

(1) Number of input file names specified

- Because a load module file is created from one or more input files, LK17K supports the specification of one or more input file names.
- Up to 128 input files can be specified. If more than 128 files are specified, however, an error message is output and LK17K stops processing.

(2) Specification of an invalid input file name

- If a specified input file name does not exist, an abort error will occur.
- If a specified input file name consists only of a device type file name or path name, an abort error will occur.

(3) Specification of duplicate input file names

- If the same input file is specified more than once, an abort error will occur.
- If a specified input file name is the same as a parameter file name, an abort error will occur.
  Even when the names specified in the command line appear to be different (for example, when one is specified with a path name while the other is specified without a path name), an abort error will occur if they are ultimately found to be the same once path names have been added to the names (the names indicate the same entity).

2.3.4 Specifying Output Files

(1) Specifying an output file name

- Specify an output file name in an option.
- An output file name can be specified with a path name only.
  When only a path name is specified, an output file is created with the specified path name plus an appropriate default name. (For an explanation of default names, see **Section 2.3.5**.)
  When a path name consists only of directory names, the last "\" can be omitted.

(2) Specification of an invalid output file name

- If a specified output file already exists as a read only file, an abort error will occur. If the file is other than a read only file, the file will be overwritten.
- If device type file name CLOCK (clock) is specified as the output destination of an ASCII file, an abort error will occur. An abort error will also occur if CON (con), PRN (prn), NUL (nul), or CLOCK (clock) is specified as the output destination for a binary file.
- If a specified output file name contains an invalid path name, an abort error will occur.

(3) Specification of duplicate output file names

- If a specified output file name is the same as an input file name, another output file name, or parameter file name, an abort error will occur.
  Even when the names specified in the command line appear to be different (for example, when one is specified with a path name while the other is specified without a path name), an abort error will occur if they are ultimately found to be the same once path names have been added to the names (the names indicate the same entity).

2.3.5 Interpreting Output File Names

Specify an output file name in an output file name change option.

Table 2-3. Interpreting Output File Names

Option / Processing	When an output file name is changed			When an output file name is not changed
	A file name is specified	Only a path name is specified	A device name is specified	
Processing performed by LK17K	Specified file name (plus an appropriate default extension if omitted from the specified file name)	Specified path name plus the first specified input file name, the extension of which has been changed to an appropriate default extension	Specified device name	Current path name plus the first specified input file name, the extension of which has been changed to an appropriate default extension

Remark For an explanation of the default extensions, see **Section 2.3.6**.

Table 2-4. Examples of Output File Name Specification

Example	Description in the command line	Input file name	Output file name
1	X>LK17K -OBJ = OUT.LNK IN1.REL IN2.REL	IN1.REL, IN2.REL	OUT.LNK, IN1.ICE
2	X>LK17K -OBJ = \LM IN1.REL IN2.REL	IN1.REL, IN2.REL	If path name “\LM” is specified \LM\IN1.LNK, IN1.ICE
			If path name “\LM” is not specified \LM.LNK IN1.ICE
3	X>LK17K -OBJ = \LM\ IN1.REL IN2.REL	IN1.REL, IN2.REL	If path name “\LM\” is specified \LM\IN1.LNK, IN1.ICE
			If path name “\LM\” is not specified, an abort error will occur.
4	X>LK17K -OBJ = NULL IN1.REL IN2.REL	IN1.REL, IN2.REL	NULL, IN1.ICE
5	X>LK17K IN1.REL IN2.REL	IN1.REL, IN2.REL	IN1.LNK, IN1.ICE

2.3.6 Default Extensions

The following default extensions are assumed if an input files are specified without extensions.

Table 2-5. Default Extensions

File type	Default extension
Object module file	.REL
Directive file	.DR
Parameter file	.PLK
Load module file (.ICE)	.ICE
Load module file (.PRO)	.PRO
Link object module file	.LNK
Link map file	.LMP

2.4 TEMPORARY FILES

LK17K creates several temporary files during processing.

Temporary files are automatically deleted when LK17K terminates (normally or abnormally) or the execution of LK17K is interrupted using the CTRL-C key combination.

Even when a file having the same name as a temporary file already exists, the file will be overwritten unless it is write-protected.

Temporary files are created in the drive and directory determined according to the following priority.

<1> Drive and directory specified in the work path specification option (-WORK)

<2> Drive and directory specified by environment variable TMP

<3> Current drive and directory

The following table lists temporary files.

Table 2-6. Temporary Files

Temporary file name	Explanation
LKxxxxxx	Temporary files used by LK17K for symbol processing
LKyyyyyy	
LKzzzzzz	
LKmmmmmm.ICE	Temporary files used by LK17K for output file creation processing
LKmmmmmm.PRO	
LKmmmmmm.DIF	
LKmmmmmm.LMP	
LKnnnnnn	Temporary file used by LK17K for an input file

Remark mmmmmm, nnnnnn, xxxxxx, yyyyyy, and zzzzzz are arbitrary numeric strings.

* 2.5 NUMBER OF SYMBLOS

When the symbol and section names consist of eight characters each (four characters when in Japanese), and there is at least 460K bytes of free memory space on the host machine, the number of symbols will be as follows:

Total number of symbols: Public symbols : 65535
Local symbols : The total number of symbols output by an assembler is guaranteed.

2.6 ENVIRONMENT VARIABLE

The following explains the environment variable which can be used with LK17K.

- TMP: Environment variable for specifying the path in which temporary files are to be created.

It is recommended that a high-speed file device, such as a RAM disk, be specified for environment variable TMP.

2.7 INTERRUPTING PROCESSING

Entering the CTRL-C key combination returns control to the OS. In this case, all open temporary files are deleted.

[MEMO]

CHAPTER 3 OPERATION

3.1 STARTUP

3.1.1 Startup with Options Specified in the Command Line

[Format]

X>[path-name]LK17KΔ [-option[Δ-option...]]Δ<object-module-file-name>[Δ-option [Δ-option...]]

| | | | | | |

<1> <2> <3> <4> <5> <4>

<1> Current drive name

<2> Directory name

<3> LK17K execution file name

<4> Option specifying the details of LK17K operation.

To specify more than one link option, separate the link options with blanks.

Link options may appear anywhere, either before or after the object module file name.

<5> Name of the object module file to be input to LK17K

Remark A help message appears if only LK17K is specified.

3.1.2 Startup Using a Parameter File

A parameter file is a file containing the information needed to startup LK17K, such as the input/output file names and options.

A parameter file is used if the information needed for startup is too long to be specified on the command line or if the same options are used every time LK17K is started.

To use a parameter file, specify the -PAR parameter file specification option in the command line.

[Format]

X>LK17K [Δ<object-module-file>]Δ-PAR = <parameter-file-name>

| |

<1> <2>

<1> Parameter file specification option

<2> Parameter file name

The description of a parameter file is governed by the following rules:

(1) When an object module file name is not specified on the command line, it must be specified in a parameter file.
(2) An object module file name can be specified after -PAR on the command line.
(3) A parameter file must specify all the link options and output file names which would otherwise be specified on the command line.
(4) A parameter file can contain comments.
All characters following a ";" or "#", up to the subsequent CR or EOF, are interpreted as being a comment.

[Description example]
TEST.PLK

```
; Link options
-OBJ -ICE -LMAP -MP
; input files
A.REL B.REL C.REL
```

3.1.3 Execution Start Messages

When LK17K starts, it first issues the following execution start messages to the output device, then starts linking.

```
X>[path-name]LK17KΔ[-option[Δ-option...]]Δ<object-file-list>[Δ-option[Δ-option...]]
                                                          <– Command line

17K Series Linker Vx.yz   [DD MMM YY]                     <– Execution start message
    Copyright (C) NEC Corporation XXXX                    <– Same as above
    --- Link start hh:mm:ss   yy/dd/mm ---                <– Same as above
```

x.yz	:	Version number
DD MMM YY	:	Date of creation
hh:mm:ss yy/dd/mm	:	Link start time and date

3.1.4 Help Message

When LK17K is started with no parameters specified, the following help message is issued to the output device.

```
X> [path-name]LK17K [△]                                          <– Command line

17K Series Linker Vx.yz    [DD MMM YY]
    Copyright (C) NEC Corporation XXXX

usage: LK17K [option [...] ] input-file [...]  [option [...]  ]

The option is as follows  ( [ ] means omissible).

-OBJ[ECT] [=file]/-NOOB [JECT]    :Create object module file
                                                [with specified name]/Not.
-ICE [=file]/-NOIC[E]             :Create ice file [with specified name]/Not.
-PROM [=file]/-NOP[ROM]           :Create pro file [with specified name]/Not.
-HOS[T]/-NOH[OST]                 :Use SIMPLEHOST/Not.
-WOR[K]=path-name                 :Set temporary directory.
-WAR[NING]=n                      :Set warning level.
-INC[REMENTAL]/-NOIN[CREMENTAL]   :Create dif file/Not.
-DIR[ECTIVE]=file                 :Read directive file from specified file.
-PAR[AMETER]=file                 :Read parameter file from specified file.
-PROG="prog-name"                 :Set program name.
-IND[IRECT]=section-name          :Indirect locate.
-JUN[K]/-NOJ[UNK]                 :Create object module file
                                   if fatal error occurred/Not.
-LMA[P] [=file]/-NOL[MAP]         :Create link map file [with specified name]/Not.
-ML/-NOML                         :Output local symbol list to link map file/Not.
-MP/-NOMP                         :Output public symbol list to link map file/Not.

DEFAULT ASSIGNMENT: -OBJ -ICE -NOP -NOHOST -WAR0 -NOIN -NOJUNK -LMA
                    -NOML -NOMP

directive file usage :
MERGE section-name:[location-type-definition]  [merge-type-definition]
      [=segment-name]

example : MERGE SEC1 : BOOT
          MERGE SEC2 : SBR SEQUENT = SEG0
X>                                                               <- OS prompt
```

Help message

3.1.5 Terminating the Program

LK17K outputs the normal termination message when it terminates normally, or the abnormal termination message if an abort error occurs.

The normal termination messages and abnormal termination message are shown below.

(1) Normal termination messages

```
X> [path-name] LK17KΔ [-option [Δ-option...] ]Δ <object-module-file> [Δ-option
[Δ-option...] ]

17K Series Linker Vx.yz  [DD MMM YY]
  Copyright (C) NEC Corporation XXXX

--- Link start hh:mm:ss  yy/dd/mm  ---
--- Link end   hh:mm:ss  yy/dd/mm  ---                <– Normal termination message

Device file name : MMMMMMMM.DEV  (Version:NN)

Total error (s) : XXXXX Total warning (s) : YYYYY   <– Normal termination message

X>                                                  <– OS prompt
```

MMMMMMMM : Device file name
NN : Device file version number
XXXXX : Number of errors (in decimal)
YYYYY : Number of warnings (in decimal)

(2) Abnormal termination message (output when a specified file cannot be found)

```
X> [path-name] LK17KΔ [-option [Δ-option...] ]Δ<object-module-file> [Δ-option
[Δ-option...] ]

17K Series Linker Vx.yz     [DD MMM YY]
  Copyright (C) NEC Corporation XXXX

--- Link start hh:mm:ss  yy/dd/mm ---
error A003 : File ‘xxxxx’ is not found
--- Link end hh:mm:ss yy/dd/mm ---

Program aborted                                     <–Abnormal termination message

X>                                                  <–OS prompt
```

3.1.6 Error Levels

LK17K returns one of the following values to the MS-DOS errorlevel to indicate the termination state.

Table 3-1. Error Levels

Termination state	errorlevel
Normal termination	0
Warning	0
Fatal error	1
Abort error	2

3.2 INPUT

LK17K can input any of the following as input:

- Object module file (.REL)
- Link option
- Directive file
- Parameter file

Link options and an object module file name may be specified on the command line or in a parameter file.
A directive file name is specified with the -DIR directive file name specification option.
A parameter file name is specified with the -PAR parameter file name specification option.

3.2.1 Object Module File

LK17K can input the object module file (.REL) produced by the assembler (RA17K) or compiler (*emIC-17K*).

3.2.2 Link Options

A link option specifies the information necessary for linking. If the description of an option contains an error, LK17K issues an error message and stops processing.

A link option is not case-sensitive. For details, see **Section 3.2.3**.

(1) Specification

An option can be specified in either of two ways:

- On the command line.
- In a parameter file.

(2) Description format

For an explanation of how to specify an option on the command line, see **Section 3.1.1**.
For an explanation of how to specify an option in a parameter file, see **Section 3.1.2**.

3.2.3 Link Option Types

The table below lists link options. When conflicting options are specified, the option last specified becomes valid.

Table 3-2. Link Options

Option	Explanation	Default	Interpretation when -HOST is specified	Reference page
-OBJ[ECT] [= <file-name>] -NOOB[JECT]	Link object module (.LNK) output control	-OBJ	-OBJ is forcibly assumed. (The specification is ignored.)	48
-ICE [= <file-name>] -NOIC[E]	Load module (.ICE) output control	-ICE	-ICE is forcibly assumed. (The specification is ignored.)	48
-PROM [= <file-name>] -NOP[ROM]	Load module (.PRO) output control	-NOP	The specification is valid.	49
-HOS[T] -NOH[OST]	*SIMPLEHOST* information control	-NOH	-HOST	50
-WOR[K] = <path-name>	Work path (drive and directory) specification	None	The specification is valid.	51
-WAR[NING] = n (0 ≤ n ≤ 15)	Warning output prohibition specification	-WAR = 0	The specification is valid.	51
-INC[REMENTAL] -NOIN[CREMENTAL]	Differential file output control	-NOIN	The specification is valid.	52
-DIR[ECTIVE] = <file-name>	Directive file specification	None	The specification is valid.	53
-PAR[AMETER] = <file-name>	Parameter file specification	None	The specification is valid.	53
-PROG = “program-name”	Program name output control	None	The specification is valid.	54
-IND[IRECT] = <section-name>	Indirect allocation specification	None	The specification is valid.	54
-JUN[K] -NOJ[UNK]	Link object module (.LNK) forcible output control -JUNK is ignored when -NOOB is specified.	-NOJ	The specification is valid.	55
-LMA[P] [= <file-name>] -NOL[MAP]	Link map file (.LMP) output control	-LMA	The specification is valid.	56
-ML -NOML	Control of output of local symbol list to link map file -ML is ignored when -NOL is specified.	-NOML	The specification is valid.	56
-MP -NOMP	Control of output of public symbol list to link map file -MP is ignored when -NOL is specified.	-NOMP	The specification is valid.	57

Remark When conflicting options are specified, the option specified last becomes valid. This also applies when options are specified in a parameter file. The following explains the processing performed when both the command line and a parameter file are used.

Example -OBJ is specified on the command line and -NOOB is specified in parameter file EX.PLK

<1> If LK17K -OBJ -PAR=EX, -NOOB is valid.

<2> If LK17K -PAR=EX -OBJ, -OBJ is valid.

(1) Link object module file (.LNK) output control

[Format]

-OBJ [ECT] [= <file-name>]
-NOOB [JECT]
(default: -OBJ [ECT])

[Function]

These options control the output of a link object module file (.LNK).

[Explanation]

(1) -OBJ [ECT] [= <file-name>]
Specifies that a .LNK file should be output.
<file-name> specifies the name of the .LNK file.
<file-name> can contain a path name (drive and directory names).
If -OBJ is specified without <file-name>, the default described in (3) is assumed. If only a path name is specified in <file-name>, the path name plus the file name described in (3) is assumed.

(2) -NOOB [JECT]
A .LNK file is not output.

(3) When no options are specified (default)
<1> If there is more than one input object module file (.REL)
Output destination: Current path
File name: <first-input-object-module-file-name> + extension (.LNK)
<2> If there is only one input object module file (.REL)
Output destination: Current path
File name: <object module file name> + extension (.LNK)

[Relationship with other options]

When -HOST is specified, -OBJ=<file-name> and -NOOB are ignored, -OBJ being assumed.

(2) Load module file (.ICE) output control

[Format]

-ICE [= <file-name>]
-NOIC [E]
(Default: -ICE)

[Function]

These options control the output of a load module file (.ICE).

[Explanation]

(1) -ICE [= <file-name>]

Specifies the output of a .ICE file.

<file-name> specifies the name of the .ICE file.

<file-name> can contain a path name (drive and directory names).

If -ICE is specified without <file-name>, the default described in (3) is assumed. If only a path name is specified in <file-name>, the path name plus the default file name described in (3) is assumed.

(2) -NOIC [E]

A .ICE file is not output.

(3) When no options are specified (default)

<1> When there is more than one object module file

Output destination: Current path

File name: <first-input-object-module-file-name> + extension (.ICE)

<2> When there is only one object module file

Output destination: Current path

File name: <object-module-file-name> + extension (.ICE)

[Relationship with other options]

(1) When -HOST is specified, -ICE=<file-name> and -NOIC are ignored, -ICE being assumed.

(2) -ICE and -NOIC do not affect the other options.

(3) Load module file (.PRO) output control

[Format]

-PROM [= <file-name>]

-NOP [ROM]

(Default: -NOP [ROM])

[Function]

These options control the output of a load module file (.PRO).

[Explanation]

(1) -PROM [= <file-name>]

Specifies that a .PRO file should be output.

<file-name> specifies the name of the .PRO file.

<file-name> can contain a path name (drive and directory names).

If -PROM is specified without <file-name>, a .PRO file is output according to the following rules:

<1> When there is more than one object module file

Output destination: Current path

File name: <first-input-object-module-file-name> + extension (.PRO)

<2> When there is only one object module file
Output destination: Current path
File name: <object module file name> + extension (.PRO)
When only a path name is specified in <file-name>, a file is output with the path name and file name determined according to the above rules.

(2) -NOP [ROM]
A .PRO file is not output.

(3) When no options are specified (default)
-NOP is assumed.

[Relationship with other options]
-PROM and -NOP do not affect the other options.

(4) *SIMPLEHOST* information control

[Format]
-HOS [T]
-NOH [OST]
(Default: -NOH [OST])

[Function]
These options control the output of the information needed to use the 17K series *SIMPLEHOST*.

[Explanation]
(1) -HOS [T]
Specify this option when *SIMPLEHOST* is used.

(2) -NOH [OST]
Specify this option when *SIMPLEHOST* is not used.

(3) When no options are specified (default)
-NOH is assumed.

[Relationship with other options]
(1) -NOHOST does not affect the other options.

(2) -HOST affects the following options.
-ICE=<file-name> and -NOIC -> -ICE (no file name specified) is assumed.
-OBJ=<file-name> and -NOOB -> -OBJ (no file name specified) is assumed.

[Notes]
(1) When -HOST is specified at link time, an error will occur if at least one input object module file (.REL) has been assembled without specifying -HOST.

(2) When an object module file (.REL), assembled without specifying -HOST, is specified as an input file, an error does not occur even if -HOST is not specified at link time. In this case, however, *SIMPLEHOST* cannot be used.

(5) Work path control

[Format]

-WOR [K] = <path-name>

[Function]

This option specifies the name of a path (drive/directory names) in which the temporary files used during linking are to be stored.

-WORK takes precedence over the environment variable TMP.

[Explanation]

(1) Only one drive name can be specified.

(2) When only a drive name is specified in <path-name>, temporary files are created in the current directory of the specified drive.

(3) When a drive name is specified, followed by a directory name, temporary files are created in the specified directory.

(4) When only a directory name is specified in <path-name>, temporary files are created in the specified directory of the current drive.

(5) The work path is determined according to the following priority:

<1> Drive and directory specified in the work path specification option (-WORK)

<2> Drive and directory specified in the environment variable TMP

<3> Current drive and directory

(6) It is recommended that a high-speed file device, such as RAM disk, be specified in the work path.

[Relationship with other options]

-WORK does not affect the other options.

[Notes]

(1) An error will occur if the specified drive or directory does not exist.

(2) All temporary files are deleted once linking has been completed.

(6) Warning output prohibition control

[Format]

-WAR [NING] = n

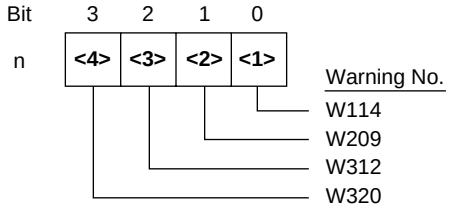
(Default: -WAR = 0)

[Function]

This option specifies whether warnings should be output during linking.

[Explanation]

(1) Warnings are not output if the corresponding bits are set to “1” with numeric value n expressed in binary.



(2) Numeric value n can be specified in binary, decimal, or hexadecimal.

[Specification example]

-WAR = 2: Warning W209 is not output.

[Relationship with other options]

-WAR does not affect the other options.

[Notes]

An error will occur if numeric value n is invalid (n < 0 or 15 < n).

(7) Differential file output control

[Format]

-INC [REMENTAL]
-NOIN [CREMENTAL]
(Default: -NOIN [CREMENTAL])

[Function]

These options control the output of a differential file (.DIF) for incremental load.
For an explanation of incremental load, see **Section 2.6** in **Part I**.

[Explanation]

(1) -INC [REMENTAL]
A differential file containing the differences between the new and old .ICE files is output.
(2) -NOIN [CREMENTAL]
A differential file is not output.
(3) If no options are specified (default)
-NOIN is assumed.

[Relationship with other options]

-INC and -NOIN do not affect the other options.

[Notes]

(1) When -INC is specified but a .LNK file cannot be found, an error occurs and linking stops.

(2) An error will occur if -INC is specified together with -NOOB or -NOIC.

(8) Directive file specification

[Format]

-DIR [ECTIVE] = [<path-name>] <file-name>

[Function]

This option causes a specified file to be input as a directive file.
For details of directive files, see **Section 2.1.2** in **Part I**.

[Explanation]

<file-name> specifies the name of a file to be used as a directive file.
<file-name> can contain a path name (drive/directory name).
When the extension is omitted, .DR is assumed.

[Relationship with other options]

-DIR does not affect the other options.

[Notes]

(1) An abort error will occur if a file which does not exist is specified.

(2) An error will occur if <file-name> is not specified.

(9) Parameter file specification

[Format]

-PAR [AMETER] = [<path-name>] <file-name>

[Function]

This option specifies that options and an input file name should be entered from a parameter file.
The functions of this option are as follows:

- Specify this option when the information needed to start LK17K cannot be specified in the space available on the command line.
- When the same options are to be used every time linking is performed, specify these options in a parameter file and specify the parameter file with this option.

[Explanation]

(1) <file-name> specifies the name of a file to be used as a parameter file.
<file-name> can contain a path name (drive/directory name).
When the extension is omitted, .PLK is assumed.
An abort error will occur if a file which does not exist is specified.
(2) An abort error will occur if no file name is specified.
(3) Parameter files cannot be nested. An abort error will occur if -PAR is specified in a parameter file.
(4) All characters following a ";" or "#", up to the subsequent CR or EOF, are handled as a comment.
(5) An abort error will occur if -PAR is specified more than once.

[Relationship with other options]

-PAR does not affect the other options.

(10) Program name output control

[Format]

-PROG = "program-name"

[Function]

This option outputs a specified character string (program name) to a load module file.
The character string is output to each of the program name fields in the absolute address list, local cross reference list, public cross reference list, map list, and report list produced by the document processor (DOC17K).

[Explanation]

(1) Up to 127 characters can be specified for <program-name>. However, only the first 32 bytes are output to the load module file.
An error will occur and linking will stop if 128 or more characters are specified.
(2) If this option is omitted, nothing is output to the program name fields.

[Relationship with other options]

-PROG does not affect the other options.

(11) Indirect allocation specification

[Format]

-IND [IRECT] = <section-name>

[Function]

This option indirectly allocates the section block for which the relocation address VECTn (n = max) is specified.

[Explanation]

(1) By default, the section block for which the relocation attribute VECTn (n = max) is specified is directly allocated to address n (PAGE0) of segment 0. When -IND is specified, a BR addr instruction is created at address n of PAGE0 in segment 0 and the section block having the relocation attribute VECTn (n = max) is allocated to any address in segment 0 (indirect allocation).
Specifying -IND, thus increasing the free area in PAGE0 of segment 0, allows more subroutines to be allocated within PAGE0 of segment 0, because subroutine branch destinations are always allocated within PAGE0 of segment 0.

(2) If -IND is specified, the object code will become larger because a BR addr instruction is created.

[Relationship with other options]

-IND does not affect the other options.

(12) Link object file forcible output control

[Format]

-JUN [K]
-NOJ [UNK]
(Default: -NOJ [UNK])

[Function]

The JUN option forces a link object file (.LNK) to be output even if a fatal error occurs. Note, however, that a file is not output if -NOOB is specified at the same time.

[Explanation]

(1) -JUN [K]
A link object file is output even if a fatal error occurs. When -NOOB is specified, -JUNK is ignored.

(2) -NOJ [UNK]
A link object file is not output if a fatal error occurs.

(3) When no options are specified (default)
-NOJ is assumed.

[Relationship with other options]

-JUN is ignored when -NOOB is specified.

(13) Link map file (.LMP) output control

[Format]

-LMA [P] [= <file-name>]
-NOL [MAP]
(Default: -LMA [P])

[Function]

These options control the output of a link map file (.LMP).

[Explanation]

(1) -LMA [P] [= <file-name>]
Specifies that a .LMP file should be output.
<file-name> specifies the directory or the name of the .LMP file.
If -LMA is specified without <file-name>, the default described in (3) is assumed. If only a path name is specified for <file-name>, the path name plus the default file name described in (3) is assumed.

(2) -NOL [MAP]
A link map file is not output.

(3) When no options are specified (default)
<1> If there is more than one object module file
Output destination: Current path
File name: <first-input-object-module-file-name> + extension (.LMP)
<2> If there is only one object module file
Output destination: Current path
File name: <object-module-file-name> + extension (.LMP)

[Relationship with other options]

-LMA and -NOL do not affect the other options.

[Notes]

<file-name> for -LMA may contain a path name (drive/directory name).

(14) Local symbol list output control

[Format]

-ML
-NOML
(Default: -NOML)

[Function]

These options control the output of a local symbol list.
A local symbol list is output to a link map file (.LMP).

[Explanation]

(1) -ML
A local symbol list is output to a .LMP file.

(2) -NOML
A local symbol list is not output to a .LMP file.

(3) When no options are specified (default)
-NOML is assumed.

[Relationship with other options]

When -NOL is specified, -ML is ignored.

(15) Public symbol list output control

[Format]

-MP
-NOMP
(Default: -NOMP)

[Function]

These options control the output of a public symbol list.
A public symbol list is output to a link map file (.LMP).

[Explanation]

(1) -MP
A public symbol list is output to a .LMP file.

(2) -NOMP
A public symbol list is not output to a .LMP file.

(3) When no options are specified (default)
-NOMP is assumed.

[Relationship with other options]

When -NOL is specified, -MP is ignored.

3.3 OUTPUT

LK17K outputs the following:

- Messages
  - Informational messages
    - Execution start messages
    - Help message
    - Termination messages
  - Error messages

- Output files
  - Load module file (.ICE)
  - Load module file (.PRO)
  - Link object module file (.LNK)
  - Log file (LK17K.LOG)
  - Differential file (.DIF)
  - Link map file (.LMP)

3.3.1 Output Messages

LK17K outputs two types of messages:

- Informational messages
- Error messages

(1) Informational messages

There are three types of informational messages:

- Execution start messages
- Help message
- Termination messages

(a) Execution start messages

These messages are output to notify the user that linking has started.
The message formats are given in **Section 3.1.3**.

(b) Help message

This message is output if LK17K is started with no parameters specified.
It explains how to use LK17K and presents a brief explanation of the supported options. The format of the help message is given in **Section 3.1.4**.

(c) Termination messages

These messages are output to notify the user that LK17K has terminated.

There are two types of termination messages: normal termination messages and the abnormal termination message.

(i) Normal termination messages

These messages are output when linking has been completed successfully.

The formats of these messages are given in **Section 3.1.5**.

(ii) Abnormal termination message

This message is output when LK17K cannot be executed for any of the following reasons:

- A specified file name or option has incurred a fatal error.
- A fatal file I/O error has occurred.
- An error occurred during LK17K processing.

The format of this message is given in **Section 3.1.5**.

(2) Error messages

An error message is output in the following format. Each error message is also issued to the log file (LK17K.LOG).

[Format]

<1> For an error in a directive file

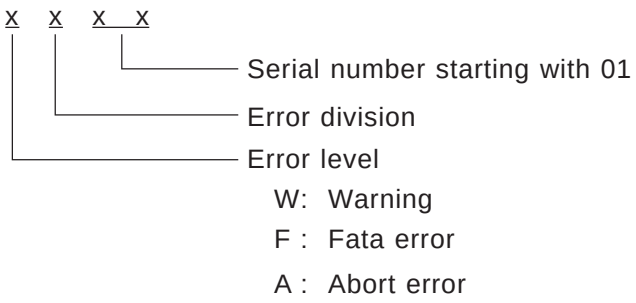
```
Input file name (line number)Δ : Δ Error number Δ : Δ Error message
```

<2> For another error

```
errorΔ Error number Δ : Δ Error message
```

[Explanation]

- Input file name
  - A file name entered in lower case is converted to upper case. (Japanese-language characters are not converted.)
  - An input path name is output exactly as specified. (Default names are not output.)
  - If an include file contains an error, the include file name and the number of the line containing the error are output.
- Line number
  - Line numbers are left-justified when output.
- Error number
  - Error numbers are indicated using four characters.



- Error messages
  For details, see **Chapter 4**.

3.3.2 Output Files

LK17K outputs the following files:

- Load module file (.ICE): For loading the in-circuit emulator
- Load module file (.PRO): For receiving code and writing OTP products
  .ICE and .PRO files are output as executable load modules.
  The .ICE and .PRO files conform to the extended Intel HEX format.
  LK17K checks the versions of the tools (*emIC-17K*, RA17K, and LK17K) and the device files it uses prior to outputting a load module file. If any of the tools is not of the correct version, warning W115 is output.
- Link object module file (.LNK)
  This file stores the linking results in binary format. It is used as the input file for the document processor (DOC17K) and *SIMPLEHOST*.
- Log file (LK17K.LOG)
  This file stores the messages output to the screen during execution. It also stores the link start and end times.
- Differential file (.DIF)
  This file contains the differences between the new and old .ICE files used during the execution of incremental load.
- Link map file (.LMP)
  This file stores allocation information for each section, a local symbol list, and public symbol list after linking.

(1) Output format of a load module file (.ICE)

A .ICE file consists of two parts:

- Object area
  If the object code of a source program exceeds the normal program area, causing the excess to be stored in a patch area, the excess is also output to this area for debugging.
  The output complies with extended Intel HEX format. Object code of up to 256K bytes (3FFFFH) is supported.
  The 17K series supports products with up to 128K bytes of ROM. (The remaining 128K bytes are used as a patch area.)

- Debug information area
  Data to be debugged by *SIMPLEHOST* or IE-17K is output to this area.

Figure 3-1 shows the output format of a .ICE file. The contents of EAR and END RECORD in the figure are as follows.

- EAR: Extended Address Record

  0 2 0 0 0 0 0 2 1 0 0 0 E C (For EAR 1000)
  <1> **<2>** **<3>** **<4>**

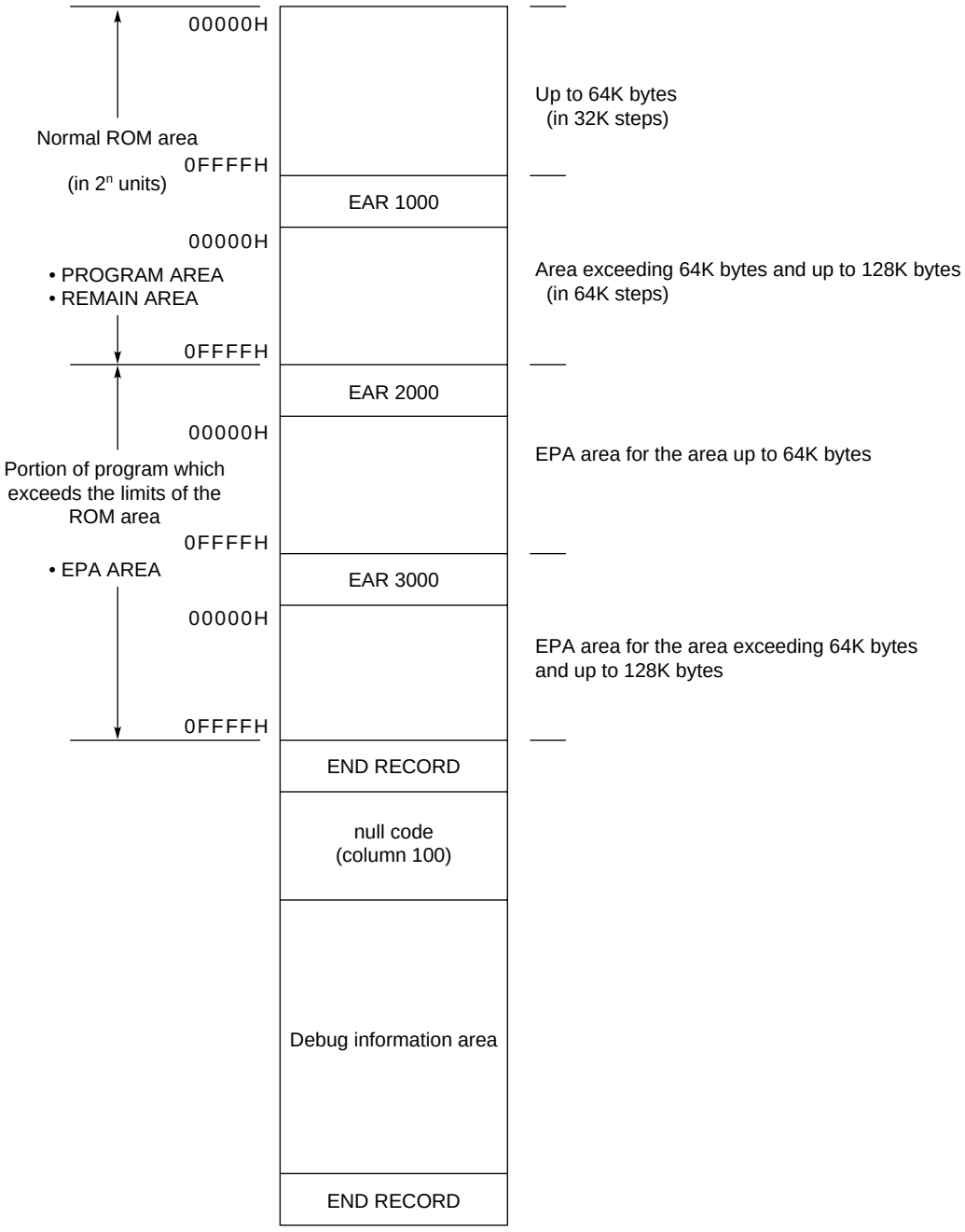
  <1>: Number of data items (in bytes)
  <2>: Record type (02 for an extended address record)
  <3>: Data (EAR data, offset value. EAR value in the figure)
  <4>: Check sum

- END RECORD: Indicates the end of the data in the object area or debug information area.

  0 0 0 0 0 0 0 1 F F
  <1> **<2><3>**

  <1>: Number of data items (in bytes)
  <2>: Record type (01 for the last record)
  <3>: Check sum

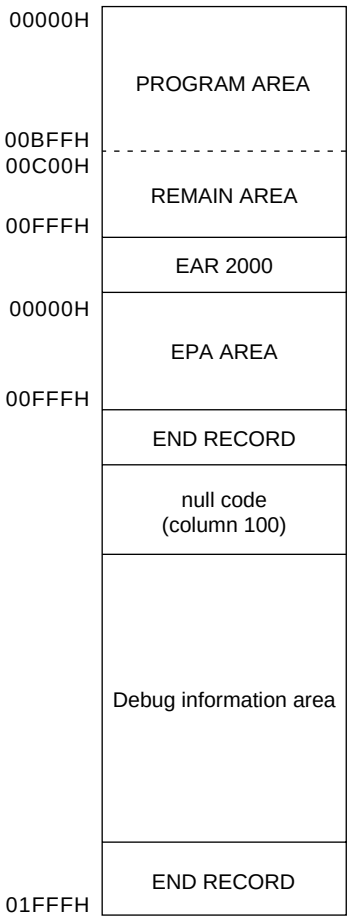
Figure 3-1. .ICE File Code Output Format



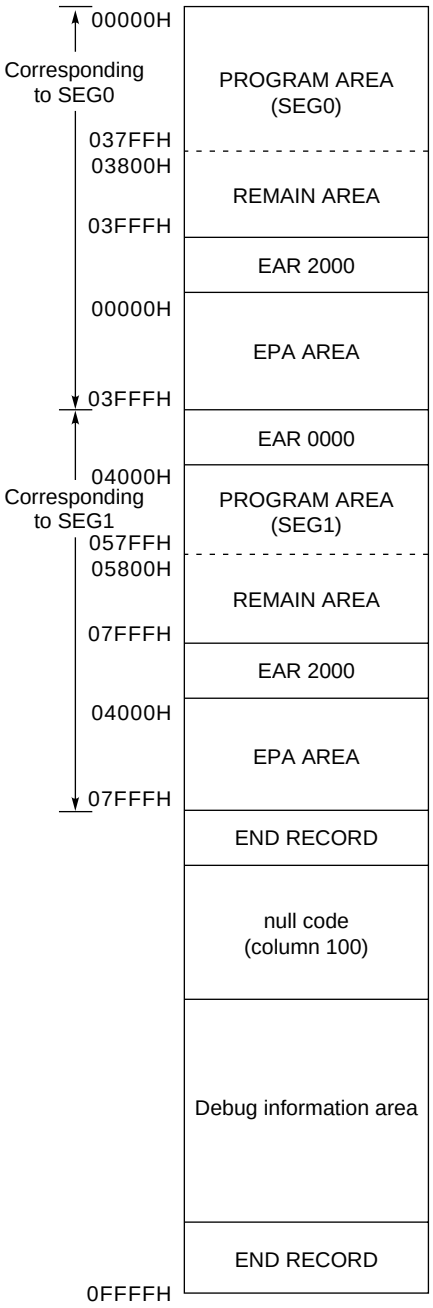
Caution The address range of a .ICE file differs with each product in the 17K series. For details, refer to the user's manual or the device file for each product.

[.ICE file output example]

Example 1. For a device whose PROGRAM area is in 1.5K steps



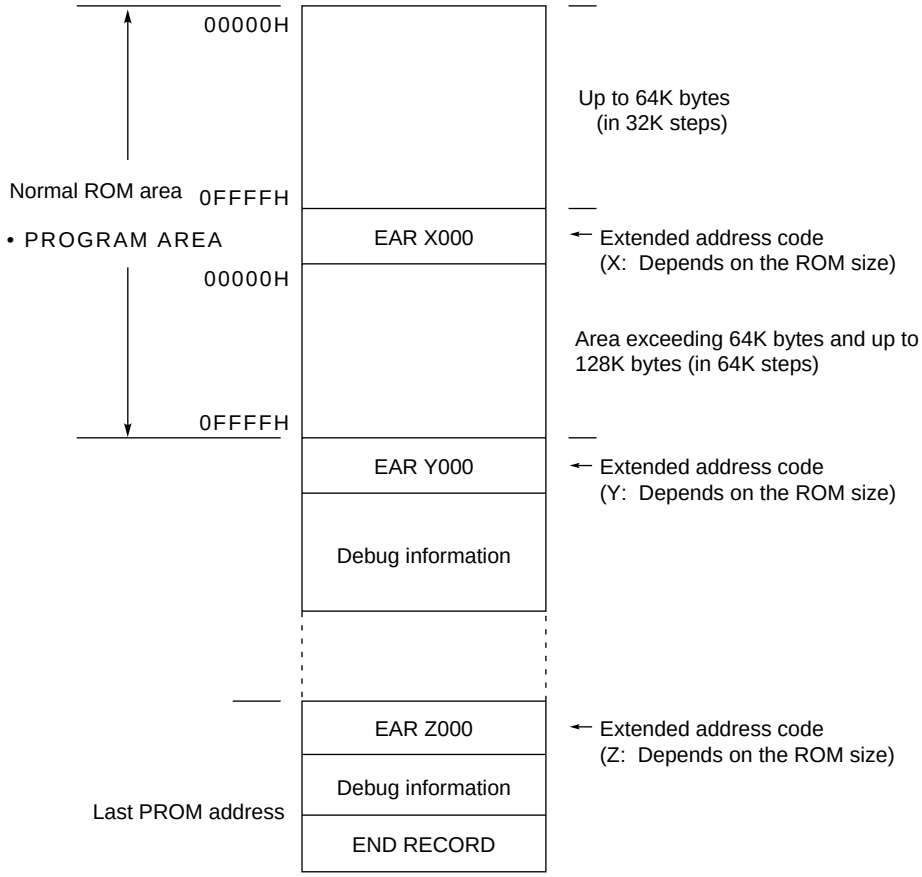
Example 2. For a device whose PROGRAM area is in 7K steps in SEG0 and in 3K steps in SEG1



(2) Output format of a load module file (.PRO)

A .PRO file is a file that complies with extended Intel HEX format, used to store object code in its program area, together with debug information. For debugging, write a .PRO file to UVPROM and mount it on the SE board. A .PRO file is also written to an OTP product when performing system debugging.

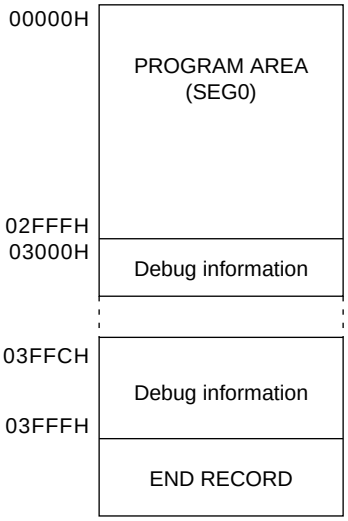
Figure 3-2. PROM File Code Output Format



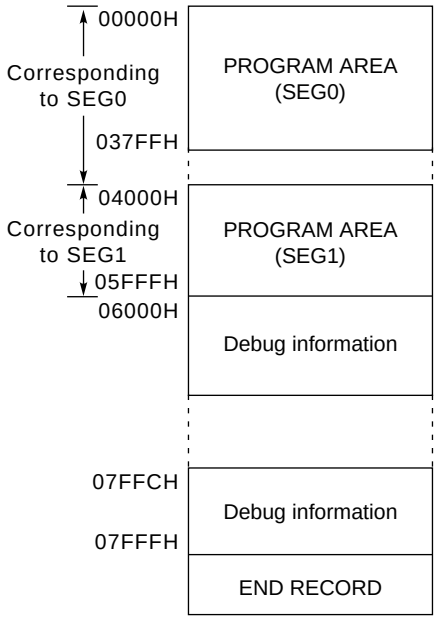
Caution The address range of a .PRO file differs with each product in the 17K series. For details, refer to the user's manual or the device file for each product.

[.PRO file output example]

Example 1. If the program area is in 6K steps (12K bytes within SEG0)



Example 2. If the program area is in 7K steps (14K bytes) in SEG0 and in 4K steps (8K bytes) in SEG1.

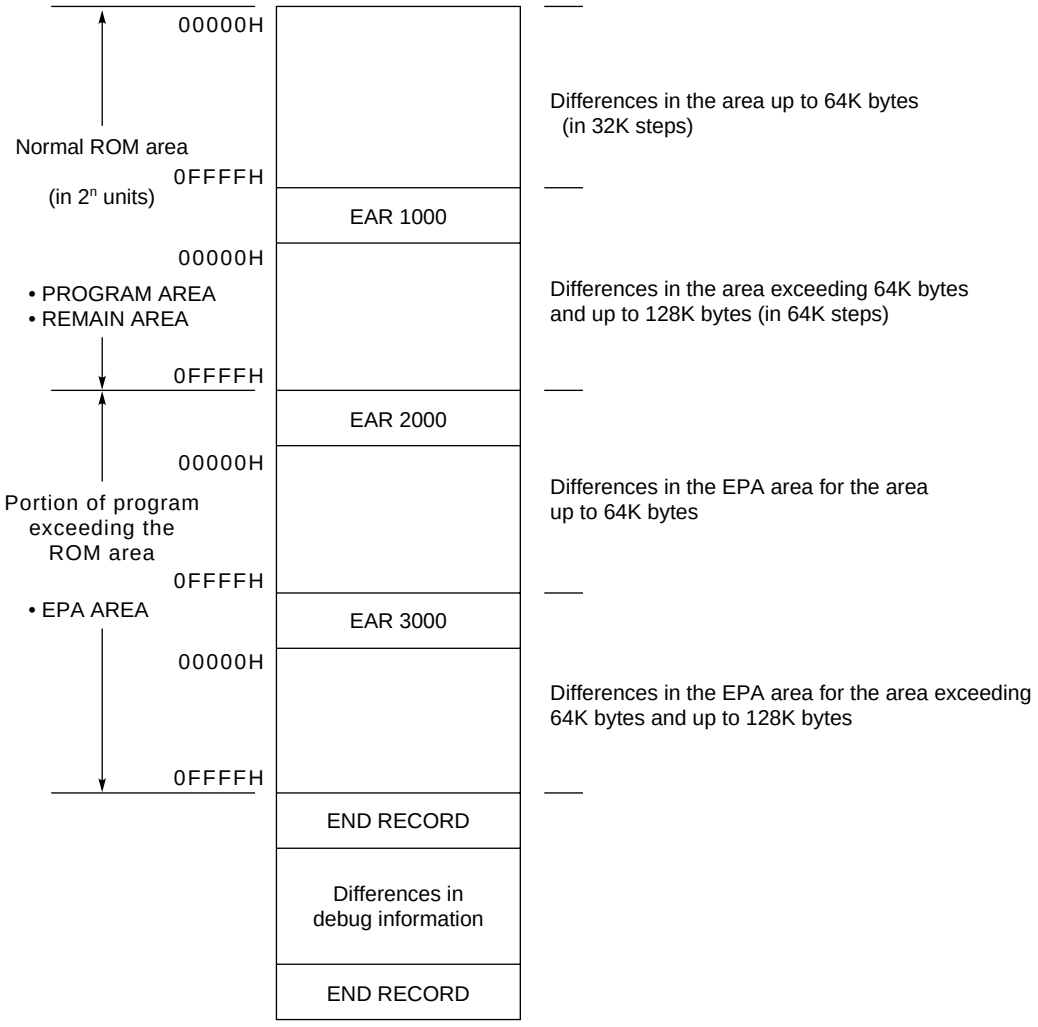


(3) Output format of a differential file (.DIF)

A differential file stores .ICE file object differences, as well as differences in debug information.

An END RECORD always appears at the end of each difference output area. This means that an END RECORD will appear even if the difference output area does not actually contain any differences. An extended address record (EAR) is also output as required.

Figure 3-3. Differential File Code Output Format



(4) Link map file

LK17K outputs a link map file to notify the user of the linking results.

A link map file consists of five parts, which are output in the following order.

(a) ID division
(b) Map list
(c) Local symbol list
(d) Public symbol list
(e) List of un-allocated sections

Table 3-3. Link Map File

Link map file	Output specification option	Default	Output order
ID division	-LMAP	Output	1
Map list	-LMAP	Output	2
Local symbol list	-ML	Not output	3
Public symbol list	-MP	Not output	4
List of un-allocated sections	-LMAP	Output	5

The following explains the layout of a link map file.

(a) Overall drawing

The following is an overall drawing of a link map file.

Figure 3-4. Overall Drawing of Link Map File

```
LK17K V1.00 <<D17xxx V1>>                              HH:MM:SS YY/MM/DD PAGE nnn

Command:       TEST1 TEST2 -ML -MP
Para-file:
Out-file:      TEST1.LNK
Map-file:      TEST1.LMP
Direc-file:
Directive:
Dev-file:      J:\D17xxx.DEV

*** Memory map ***

ROM AREA : xxxxH - xxxxH
```

OUTPUT SECTION	INPUT SECTION	INPUT MODULE	START ADDR	END ADDR	ATTR	ROM
SEC1			0H	10H	SEC	PRO
	SEC1	TEST1	0H	10H	SEC	PRO
*gap *			11H	1fffH		

```
^L

LK17K V1.00 <<D17xxx V1>>                              HH:MM:SS YY/MM/DD PAGE nnn

*** Local symbol list ***
```

MODULE	ATTR	VALUE	SYMBOL NAME
TEST1	MEM	1H	MEM1
TEST2	MEM	2H	MEM2

```
^L

LK17K V1.00 <<D17xxx V1>>                              HH:MM:SS YY/MM/DD PAGE nnn

*** Public symbol list ***
```

MODULE	ATTR	VALUE	SYMBOL NAME
TEST1	DAT	5H	DAT1

```
^L

LK17K V1.00 <<D17xxx V1>>                              HH:MM:SS YY/MM/DD PAGE nnn

*** Not allocated section ***
```

TYPE	SIZE	SECTION
AT	10H	SEC2

(b) Page

There are 66 lines per page.

A maximum of 72 characters can be specified on each line. When more characters must be displayed, they are subsequently processed.

(c) Header division

```
^L

LK17K V1.00 <<D17xxx Vn>>          HH:MM:SS YY/MM/DD PAGE nnn
```

Table 3-4. Contents of Header Division

No.	Display item	Number of digits	Display method	Contents
1	D17xxx	8 maximum	Variable digits	Displays the name of the device file being used.
2	Vn	1	Fixed digits	Displays the version of the device file being used.
3	HH:MM:SS YY/MM/DD	17	See right.	Displays the date and time when the link map file was created. HH : Hours (0 to 12, right-justified and zero-suppressed) MM : Minutes (0 to 59, right-justified and zero-suppressed) SS : Seconds (0 to 59, right-justified and zero-suppressed) DD : Day (1 to 31, right-justified and zero-suppressed) MM : Month (1 to 12, right-justified and zero-suppressed) YY : Year (94 and later, lower two digits)
4	PAGE nnn	3	Right-justified and zero-suppressed	Displays the current page number in the link map file. Initial value = 1. Incremented by 1 each time a new page is started.

<1> The specifications of each page are as follows:

(a) The top three lines are left open (blank).

(b) The above titles are displayed on the fourth line. The date and time to be displayed in No. 3 are obtained from the OS.

(c) The fifth line is left blank.

(d) A map list, etc., is output on the sixth to 63rd lines.

(e) A form feed code is output on the 64th line.

<2> When a link map, local symbol list, public symbol list, or un-allocated section list are output, a form feed code and EOF code (ascii code 1AH) are output immediately after the line feed code on the last output line.

<3> A device file name is of 8-digit variable length, left-justified.

(d) ID division

```
Command:      TEST1 TEST2 -ml -mp
Para-file:
Out-file:     TEST1.LNK
Map-file:     TEST2.LMP
Direc-file:
Directive:
Dev-file:     J:\D17xxx.DEV
```

Table 3-5. Contents of ID Division

No.	Display item	Number of digits	Display method	Contents
1	Command:	61	Left-justified	Displays the file name and options specified in the command line, each separated with a blank.
2	Para-file:			Displays the contents of a parameter file exactly as is.
3	Out-file:			Displays the name of the output file created by LK17K.
4	Map-file:			Displays the name of the link map file created by LK17K.
5	Direc-file:			Displays the name of the directive file input by LK17K.
6	Directive:			Displays the contents of the directive file exactly as is.
7	Dev-file:			Displays the name of the device file used by LK17K, as a full path name.

<1> Only a header is displayed when the corresponding display field contains no data; in this case, no blanks are output following the ":".

<2> When a display field (up to column 72) cannot contain all the data, the remainder is output on the next line, starting from column 12. Thus, when output requires two or more lines, columns 1 to 11 are filled with blanks.

<3> The input file and options specified in the command line are displayed in No. 1. Any blanks following an option in the command line are ignored.

<4> The contents of the parameter file are displayed in No. 2. Any tab contained in the parameter file is expanded to a single blank, while upper and lower case letters are output as is. Invalid characters (00H-08H, 0BH, 0CH, 0EH-19H, 1BH-1FH, and 7FH) are replaced with exclamation marks "!" before being displayed.

<5> The name of the output load module is displayed in No. 3. The name contains the name of the path used to open the load module file.

<6> The contents of the directive file are displayed in No. 6. Any tab contained in the directive file is expanded to blanks, such that the resulting characters are aligned with column 8, 16, 24, etc., relative to column 12.

<7> The device file name is displayed in No. 7. The name is displayed using its full path name to indicate the directory in which the device file is located.

(e) Memory map

```
*** Memory map ***

ROM AREA : xxxxH - yyyyH

OUTPUT SECTION   INPUT SECTION   INPUT MODULE   START ADDR   END ADDR   ATTR   ROM
SEC1                                                0H        10H     SEC    PRO
                 SEC1            TEST1              0H        10H     SEC    PRO
*  gap *                                           11H      1fffH
^L
```

Table 3-6. Contents of Memory Map

No.	Display item	Number of digits	Display method	Contents
1	ROM AREA:	4	Right-justified and zero-suppressed	Displays the start and end addresses of the device ROM.
2	OUTPUT SECTION	18	Left-justified	Displays the name of the section output by LK17K.
3	INPUT SECTION	18	Left-justified	Displays the name of the section input by LK17K.
4	INPUT MODULE	8	Left-justified	Displays the name of the module (file) input by LK17K.
5	START ADDR	6	Right-justified and zero-suppressed	Displays the start addresses (in hexadecimal) to which the sections have been allocated.
6	END ADDR	6	Right-justified and zero-suppressed	Displays the last addresses (in hexadecimal) to which the sections have been allocated.
7	ATTR	5	Left-justified	Displays the types of the sections.
8	ROM	5	Left-justified	Indicates whether the area to which the section has been allocated is the program area, EPA area, or REMAIN area.

<1> After a device file name is displayed in the ID division, two blank lines are output and the header "*** Memory map ***" is displayed on the next line, starting from column 1.

<2> One blank line is output, followed by a memory map header.

<3> After the header is displayed, one blank line is displayed, followed by the ROM area.

<4> The name of the section output by LK17K is displayed in No. 2. The name of the input section corresponding to the section name in No. 2 is displayed in No. 3. For details, see **Table 3-8**.

<5> The name of the module for the section indicated in No. 3 is displayed in No. 4.

<6> The start and end addresses to which the sections indicated in Nos. 2 and 3 are displayed in Nos. 5 and 6.

<7> The types of the sections indicated in Nos. 2 and 3 are displayed in No.7. For details, see **Table 3-7**.

<8> No.8 indicates whether the memory areas where the sections indicated in Nos. 2 and 3 are the PROGRAM or REMAIN, or EPA area. If part of a section exists in the REMAIN area, REMAIN is displayed for that section. For details, see **Table 3-9**.

Table 3-7. Section Type Display

Display	Explanation
AT	Section having the AT attribute
BOOTT	Branch table for the BOOT attribute
CROM	Section having the CROM attribute
DVECT	Section having the DVECTn attribute
VECTT	Branch table for the VECTn attribute
DSYS	Section having the DSYS attribute
SYST	Branch table for the SYS attribute
SYS	Section having the SYS attribute
DSBR	Section having the DSBR attribute
SBRT	Branch table for the SBR attribute
SBR	Section having the SBR attribute
BOOT	Section having the BOOT attribute
VECT	Section having the VECTn attribute
TABLE	Section having the TABLE attribute
SEC	Section for which no attribute is specified

Table 3-8. Section Name Display

Display	Explanation
Section name	Displays the user-specified section name. If the specified section name consists of more than 18 characters, the 18th character is replaced with an asterisk “*” and the subsequent characters are not displayed.
* gap *	* gap * is displayed for a memory area to which no section is allocated. It is displayed in No.1 only.

Table 3-9. Allocation Area Display

Display	Explanation
PRO	PROGRAM area
REM	REMAIN area
EPA	EPA area

(f) Local symbol list

```
^L

LK17K V1.00 <<D17xxx Vn>>                HH:MM:SS YY/MM/DD PAGE nnn

*** Local symbol list ***

 MODULE  ATTR   VALUE   SYMBOL NAME
 TEST1   MEM        1H  MEM1
 TEST2   MEM        2H  MEM2
```

Table 3-10. Contents of Local Symbol List

No.	Display item	Number of digits	Display method	Contents
1	MODULE	8	Left-justified	Displays the name of the input object module in which local symbols are defined.
2	ATTR	5	Left-justified	Displays the attribute of a local symbol.
3	VALUE	9	Right-justified and zero-suppressed	Displays the value (in hexadecimal) of the local symbol.**Note**
4	SYMBOL NAME	50	Left-justified	Displays the name of the local symbol.

<1> Once the map list described in the previous section has been output, a new page starts.

<2> The titles described in (c) Header division are output on the fourth line. Then, one blank line is output.

<3> "*** Local symbol list ***" is displayed, followed by one blank line, then the headers of the display items in a local symbol list.

<4> One blank line is output. A local symbol list is output starting from the tenth line. For details of how symbol attributes are displayed, see **Table 3-11**.

<5> If a symbol name consists of more than 50 characters, the 50th character is replaced with an asterisk "*", indicating continuation. The subsequent characters are not displayed.

<6> If the local symbol list is too large to fit onto one page, on the second and subsequent pages, headers are output on the sixth line, then one blank line is output, and the remainder of the symbol list is output starting from the eighth line.

Note For memory-type symbols, digits are displayed continuously, with no intervening point ".". **Example** 1.23H –> 123H

Table 3-11. Symbol Attribute Display

Display	Explanation
DAT	Data-type symbol
MEM	Memory-type symbol
LAB	Label-type symbol
FLG	Flag-type symbol

(g) Public symbol list

```
^L

LK17K V1.00 <<D17xxx Vn>>                HH:MM:SS YY/MM/DD PAGE nnn

* * * Public symbol list * * *

 MODULE  ATTR     VALUE   SYMBOL NAME
 TEST1   DAT          1H  DAT1
```

Table 3-12. Contents of Public Symbol List

No.	Display item	Number of digits	Display method	Contents
1	MODULE	8	Left-justified	Displays the name of the input object module in which public symbols are defined.
2	ATTR	5	Left-justified	Displays the attribute of a public symbol.
3	VALUE	9	Right-justified and zero-suppressed	Displays the value (in hexadecimal) of the public symbol.**Note**
4	SYMBOL NAME	50	Left-justified	Displays the name of the public symbol.

<1> After the map list described in the previous section has been output, a new page starts.

<2> The titles described in (c) Header division are output on the fourth line. Then, one blank line is output.

<3> "*** Public symbol list ***" is displayed, followed by one blank line, then the headers for the display items in a public symbol list.

<4> One blank line is output. The public symbol list is output starting from the tenth line. For details of how symbol attributes are displayed, see **Table 3-13**.

<5> If a symbol name consists of more than 50 characters, the 50th character is replaced with an asterisk "*", indicating continuation. The subsequent characters are not displayed.

<6> If the public symbol list is too large to fit onto one page, on the second and subsequent pages, headers are output on the sixth line, then one blank line is output, and the remainder of the symbol list is output starting from the eighth line.

Note For memory-type symbols, digits are displayed continuously, with no intervening point ".". **Example** 1.23H –> 123H

Caution Information output to a public symbol list relates to the definition side only; information relating to the reference side is not output.

Table 3-13. Symbol Attribute Display

Display	Explanation
DAT	Data-type symbol
MEM	Memory-type symbol
LAB	Label-type symbol
FLG	Flag-type symbol
FUNC	Label output by emIC-17K to indicate the beginning of a function

(h) Un-allocated section list

```
^L

LK17K V1.00 <<D17xxx Vn>>                HH:MM:SS YY/MM/DD PAGE nnn

* * * Not allocated section * * *

 TYPE     SIZE     SECTION
 AT          10H   SEC2
```

Table 3-14. Contents of Un-Allocated Section List

No.	Display item	Number of digits	Display method	Contents
1	TYPE	5	Left-justified	Displays the type of an un-allocated section. For details of the contents, see (f) Local symbol list.
2	SIZE	6	Right-justified and zero-suppressed	Displays the size (in hexadecimal) of the un-allocated section.
3	SECTION	59	Left-justified	Displays the name of the un-allocated section.

<1> Once the list described in the previous section has been output, a new page starts.

<2> The title line described in (c) Header division is output on the fourth line. Then, one blank line is output. The header “Not allocated section” is output on the sixth line. One blank line is output then the headers for the display items are output on the eighth line.

<3> One blank line is output. An un-allocated section list (Nos. 1-3) is output starting from the 10th line.

<4> If a section name consists of more than 59 characters, the 59th character is replaced with an asterisk “*”, indicating continuation. The subsequent characters are not displayed.

<5> If the un-allocated section list is too large to fit onto one page, on the second and subsequent pages, headers are output on the sixth line, then one blank line is output, and the remainder of the section list is output starting from the eighth line.

<6> This list is output only when an un-allocated section exists.

CHAPTER 4 ERROR MESSAGES

Error messages are classified by their numbers, as follows:

000 to 099: File specification
Option specification

100 to 199: Object module file input
Directive file input/interpretation
Preparation for linking (opening temporary files)

200 to 299: Section linking and allocation

300 to 399: Relocation solution

400 to 499: Symbol solution

500 to 599: Host machine environment

900 to 999: File I/O

Error message (A001)	"Missing input file"					
Cause	No input file has been specified.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Specify an appropriate input file.					

Error message (A002)	"Too many input files"					
Cause	The total number of input files specified exceeds the limit.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Reorganize the divided source program so that the maximum number of input files for the linker is not exceeded.					

Error message (A005)	"Illegal file specification '*filename*'"					
Cause	An illegal file name has been specified. An input file name consisting of only a device or path name cannot be specified. CLOCK cannot be specified as the output destination for an ASCII file. CON, PRN, and CLOCK cannot be specified as the output destination of a file.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the specified file name.					

Error message (A006)	"File not found '*filename*'"					
Cause	The specified input file cannot be found.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the specified path or file name.					

Error message (A007)	"Input file specification overlapped '*filename*'"					
Cause	The input file name is the same as the parameter file name.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the specified file name.					

Error message (A008)	"File specification conflicted '*filename*'"					
Cause	The same file name has been specified for multiple output files, input and output files, or parameter and output files.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Correct the input, output, and parameter file names to eliminate any duplication.					

Error message (A009)	"Unable to make file '*filename*'"					
Cause	The specified output file cannot be created because a read only file having the specified name already exists.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Specify a different name for the output file.					

Error message (A010)	"Directory not found '*filename*'"					
Cause	The output file name contains a drive or directory that does not exist.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the specified drive or directory name.					

Error message (A011)	"Illegal path '*passname*'"					
Cause	A name other than a path name has been specified in the option parameter for specifying a path name.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the path name in the option parameter. The specified path must already exist.					

Error message (A012)	"Missing parameter '*option*'"					
Cause	A necessary option parameter has not been specified.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Specify the necessary option parameter.					

Error message (A013)	"Parameter not needed '*option*'"					
Cause	An unnecessary option parameter has been specified.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Delete the unnecessary option.					

Error message (A014)	"Out of range '*option*'"					
Cause	A value that falls outside the valid range has been specified in the option parameter.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the option parameter and specify a value that falls within the valid range.					

Error message (A015)	"Parameter is too long '*option*'"					
Cause	The number of characters specified in the option parameter exceeds the limit.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Reduce the number of characters specified in the option parameter to within the limit.					

Error message (A018)	"Option is not recognized '*option*'"					
Cause	An invalid option has been specified.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Specify a valid option.					

Error message (A019)	"Parameter file nested"					
Cause	The -PAR option has been specified within the parameter file. Parameter files cannot be nested.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Delete the -PAR option from the parameter file.					

Error message (A020)	"Parameter file read error '*filename*'"					
Cause	The specified parameter file cannot be read.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check that the parameter file name has been specified correctly. Check the status of the disk on which the parameter file is stored.					

Error message (A023)	"-NOOBJ or -NOIC and -INC specified at the same time"					
Cause	The -INC option has been specified together with either the -NOOBJ or -NOIC option.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	The -INC option cannot be specified together with the -NOOBJ or -NOIC option. When specifying the -INC option, do not specify the -NOOBJ or -NOIC option.					

Error message (F102)	"Directive syntax error"					
Cause	The description of a directive contains an error.					
Program action	Ignores the directive and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Check the syntax of the directive.					

Error message (A104)	"'*filename*' Different processor type"					
Cause	The assembler or compiler type of the input object module is different from that of the initially input object module file.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Object module files using different assembler or compiler types cannot be linked. Re-assemble or re-compile object module files so that they all are of the same type.					

Error message (A106)	"Can't create temporary file '*filename*'"					
Cause	The temporary file cannot be created.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the directory name specified with the -WORK option or the TMP environment variable. Check the status of the disk on which creation of the temporary file failed (free space, medium status, etc.).					

Error message (A107)	"File '*filename*' isn't assembled with -HOST option"					
Cause	The input file was not assembled with the -HOST option (assembler) specified, even though the -HOST option (linker) was specified.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Re-assemble the affected input file with the -HOST option (assembler) specified.					

Error message (A109)	"Linker internal error"					
Cause	An internal error occurred.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Contact your authorized dealer or NEC.					

Error message (F110)	"Illegal number"					
Cause	The illegal number was found in a directive.					
Program action	Ignores the directive and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Check the number specified in the directive.					

Error message (A112)	"Section '*section*' bad vector address"					
Cause	A vector address that is not supported by the product has been specified with the VECTn/ DVECTn relocation attribute.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Specify, in the relocatable attribute, a vector address that is supported by the product.					

Error message (A113)	“Not same address (section ‘*section*’)”					
Cause	The address specified in the directive does not correspond to the segment specification.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check the correspondence between the address specified in the directive and the segment specification.					

Error message (W114)	“‘*filename*’ Different device file from first input file”					
Cause	The device file used to assemble the input file has the same name as that used to assemble the first input file, but is in a different directory.					
Program action	Issues a warning and continues processing.					
	Uses the device file used to assemble the first input file for processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	To input multiple files, use the same device file to assemble all files.					

Error message (W115)	“Can’t order ROM code with generated object code ”					
Cause	One or all of the tools used to create a load module file (assembler, compiler, and linker device file) are not of required release.					
Program action	Issues a warning and continues processing.					
	Stores version information in the item for the ACROSS check.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	To order ROM code, all the tools used must be of the required release.					

Error message (W116)	“File ‘*filename*’ file not found”					
Cause	The old .LNK file or old .ICE file cannot be found even though -INCREMENTAL has been specified.					
Program action	Issues a warning and continues processing.					
	Does not output a differential file but performs full linking.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Do not specify -INCREMENTAL when the old .LNK file or old .ICE file does not exist.					

Error message (F201)	"Multiple section definition '*section*' in merge directive"					
Cause	The section specified in a merge directive has already been catalogued. (An attempt was made to assign the same section in more than one merge directive.)					
Program action	Ignores the merge directive and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Specify a link instruction for a section in one merge directive only.					

Error message (A203)	"Section '*section*' unknown section type"					
Cause	The input object module file contains invalid section information.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check that the contents of the object module file are correct. Re-assemble or re-compile the object module file.					

Error message (A204)	"Exist same name sections"					
Cause	Merge type COMPLETE is specified for a section in a merge directive. However, one or more sections having the same name as the specified section exist in the input file.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Do not specify merge type COMPLETE for a section in a merge directive when other sections having the same name as the specified section exist.					

Error message (F206)	"Section '*section*' can't allocate to memory - ignored"					
Cause	The specified section cannot be allocated to a memory area. (There is insufficient free space in the memory area to assign the section.)					
Program action	Ignores the section and continues processing.					
	Maintains the validity of the symbol definitions in the un-allocated section.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Check that all the sections to be linked can be accommodated in the memory area. It may also prove impossible to allocate a section because of the allocation condition (absolute section etc.) of the relocation attribute of that or another section, even if the section can be accommodated in the memory area.					

Error message (F207)	"Section '*section*' has illegal section type"					
Cause	The section has been assigned an illegal section type.					
Program action	Ignores the section and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Re-assemble or re-compile the source program containing the section. Check the description of the merge directive.					

Error message (F208)	"Section '*section*' is not exist - ignored"					
Cause	The section specified in a directive does not exist.					
Program action	Ignores the section and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Do not specify a directive section that does not exist.					

Error message (W209)	"The type specified for section '*section*' is different from that specified at assemble time"					
Cause	The allocation type specified for a section in a directive differs from that specified when the section was assembled.					
Program action	Issues a warning and continues processing.					
	Regards the allocation type specified in the directive as being valid.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Do not specify a reallocation attribute for a section in a directive that differs from that specified when the section was assembled.					

Error message (F210)	"Section '*section*' can't allocate to memory (table area) - ignored"					
Cause	The section cannot be allocated to the specified table area.					
Program action	Ignores the section and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICR/PRO		---	---
User response	Check that all the sections to be linked can be accommodated in the memory area. It may also prove impossible to allocate a section because of the allocation condition (absolute section etc.) of the relocation attribute of that or another section, even if the section can be accommodated in the memory area.					

Error message (A211)	"Section '*section*' is not exist at incremental-link time"					
Cause	The section could not be found when incremental link was performed, even though it existed previously.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Do not delete any sections before performing incremental link.					

Error message (F212)	"Origin address error (file '*filename*', org addr xxxxH)"					
Cause	The address specified by the ORG dummy instruction specified in the module assembled in absolute mode is lower than the address to be assigned by the linker.					
Program action	Ignores the ORG dummy instruction and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	The addresses specified by the ORG dummy instruction in absolute mode must be in ascending order.					

Error message (W213)	"Different assemble mode"					
Cause	Object module files having different assemble modes (absolute/relocatable) were input to the linker.					
Program action	Issues a warning and continues processing. Processes the object module files of absolute mode as if they were of relocatable mode.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Do not attempt to simultaneously link object module files having different assemble modes at a time.					

Error message (A214)	"Different section type (section '*section*')"					
Cause	Sections having different allocation types are defined using the same name.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Do not define sections having different allocation types with the same name.					

Error message (F215)	"Illegal CROM segment (section '*section*')"					
Cause	The relocation attribute (CROM) and a segment are both specified. However, the CROM area does not correspond to the segment number.					
Program action	Ignores the segment and continues processing.					
File output	With -JUNK	Link object	Load module	Without -JUNK	Link object	Load module
	specification	LNK	ICE/PRO	specification	---	---
User response	When specifying both a relocation attribute (CROM) and segment number, ensure that the segment number corresponds to the CROM area.					

Error message (F301)	"Relocatable object code address out of range (file '*filename*', section '*section*', address xxxxH)"					
Cause	Modification information for the relocatable object code contained in the input object module file is output to an address where the object code does not exist.					
Program action	Ignores the relocation and continues processing.					
File output	With -JUNK	Link object	Load module	Without -JUNK	Link object	Load module
	specification	LNK	ICE/PRO	specification	---	---
User response	Check that the symbol reference is correct. Re-assemble or re-compile the file. The address displayed as address xxxxH is an absolute address subsequent to that to which the section was allocated.					

Error message (F302)	"Can't find symbol index in relocatable object code (file '*filename*', section '*section*', address xxxxH)"					
Cause	Modification information for the relocatable object code contained in the input object module file contains an error, preventing symbol information from being referenced correctly.					
Program action	Ignores the relocation entry and continues processing.					
File output	With -JUNK	Link object	Load module	Without -JUNK	Link object	Load module
	specification	LNK	ICE/PRO	specification	---	---
User response	Check that the method of referencing symbols and variables is correct. Re-assemble or re-compile the file. The address displayed as address xxxxH is an absolute address subsequent to that to which the section was allocated.					

Error message (F304)	"Operand out of range (section '*section*', address xxxxH)"					
Cause	An illegal value has been specified for an operand of an instruction.					
Program action	Issues an error and continues processing.					
File output	With -JUNK	Link object	Load module	Without -JUNK	Link object	Load module
	specification	LNK	ICE/PRO	specification	---	---
User response	In instruction operands, specify a value within the range supported by the product.					

Error message (F305)	"ROM address overflow, EPA bit on (file '*filename*', section '*section*', address xxxxH)"					
Cause	Part of the program is allocated to the EPA area or references the EPA area.					
Program action	Issues an error, sets the EPA bit for the object code to "1", then continues processing. (Object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F306)	"Out of address range [REMAIN AREA] (section '*section*', address xxxxH size yyyyH)"					
Cause	The program is larger than the available PROGRAM space and extends to the REMAIN area.					
Program action	Issues an error and continues processing. (The object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F309)	"Referring the address over area [REMAIN AREA] (file '*filename*' section '*section*', address xxxxH)"					
Cause	The program refers to an address in the REMAIN area.					
Program action	Issues an error and continues processing. (The object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F310)	"Referring the address over area [EPA AREA], EPA bit on (file '*filename*', section '*section*', address xxxxH)"					
Cause	The program references an address in the EPA area.					
Program action	Issues an error, sets the EPA bit for the object code to "1", and continues processing. (The object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (W312)	"Unreference public symbol"					
Cause	The program does not reference a symbol that is declared as being public.					
Program action	Issues a warning and continues processing.					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object LNK	Load module ICE/PRO
User response	A symbol that is not to be referenced externally must not be declared as being public.					

Error message (F313)	"Duplicated OPTION directive (file '*filename*')"					
Cause	The same mask option definition blocks are defined in multiple input files.					
Program action	Regards the first block as being valid and continues processing.					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object ---	Load module ---
User response	Delete any unnecessary mask option definition blocks.					

Error message (F314)	"Not found Mask-option block"					
Cause	For a product type having the mask option, a mask option definition block is not described in an input file.					
Program action	Issues an error and continues processing.					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object ---	Load module ---
User response	For a product type having the mask option, describe a mask option definition block.					

Error message (W315)	"Indirect addressing instructions may not work properly due to the program exceeded to EPA area"					
Cause	Because the program has reached the EPA area, it is not possible to determine whether the jump destination of an indirect branch instruction, etc., is in the PROGRAM space or in the EPA area.					
Program action	Issues a warning and continues processing.					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object LNK	Load module ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F317)	"All of the section was allocated to EPA area (file 'filename', section '*section*', address xxxxH, size yyyyH)"					
Cause	None of the sections was allocated to the PROGRAM area.					
Program action	Issues an error and continues processing. (The object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F318)	"Part of the section was allocated to EPA area (file '*filename*', section '*section*', address xxxxH, size yyyyH)"					
Cause	Part of the section cannot be allocated to the PROGRAM area.					
Program action	Issues an error and continues processing. (The object code is output correctly.)					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Reduce the size of the object code of the program.					

Error message (F319)	"Boundary error (file '*filename*', section '*section*', address xxxxH)"					
Cause	An address boundary error occurred. The object code generated by the DCP dummy instruction is allocated to an address whose lower four bits are 0FH.					
Program action	Issues an error and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Allocate the object code generated by the DCP dummy instruction to an address other than one whose lower four bits are 0FH.					

Error message (W320)	"Omitted a surplus due to an input value is over a regular value (section '*section*', address xxxxH)"					
Cause	The value specified in the DW/DB operand exceeds the specified limit.					
Program action	Issues a warning and continues processing. The surplus is truncated.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		LNK	ICE/PRO
User response	Do not specify a value that exceeds the value specified in the DW/DB operand.					

Error message (A401)	"File '*filename*' Bad symbol table"					
Cause	The symbol information in the input object module file is invalid.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Re-assemble or re-compile the file.					

Error message (A402)	"File '*filename*' has no string table for symbol"					
Cause	The symbol information in the input object module file is invalid. (No character information is found.)					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Re-assemble or re-compile the file.					

Error message (F403)	"Symbol '*symbol*' unmatched type in file *'filename1'*, First defined in file '*filename2*'"					
Cause	The type of the EXTRN/PUBLIC symbol specified in 'filename1' is different from that of the symbol having the same name as that specified in 'filename 2'.					
Program action	Ignores the type of the symbol specified in the most-recently input 'filename1' and continues processing.					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object ---	Load module ---
User response	When declaring a symbol EXTRN/PUBLIC, the type of the symbol must be the same in all the source programs for which the symbol is declared as being EXTRN/PUBLIC.					

Error message (F404)	"Multiple Symbol definition '*symbol*' in file '*filename1*', First defined in file '*filename2*'"					
Cause	The PUBLIC symbol defined in object module file 'filename1' has already been declared as being PUBLIC in object module file 'filename2'. (Duplicate PUBLIC symbols are defined.)					
Program action processing.	Ignores the symbol specified in the most-recently input 'filename1' and continues					
File output	With -JUNK specification	Link object LNK	Load module ICE/PRO	Without -JUNK specification	Link object ---	Load module ---
User response	Stop declaring either symbol as being PUBLIC or rename the symbols so that symbols having the same name are not declared as being PUBLIC in multiple source programs. (Symbols having the same name cannot be declared as being PUBLIC, irrespective of whether the symbols are of different types.)					

Error message (F405)	"Undefined symbol '*symbol*' in file '*filename*'"					
Cause	A symbol that is declared as being EXTRN in a file is not declared as being PUBLIC in another file.					
Program action	Assumes the value of the undefined symbol to be 0 and continues processing.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		LNK	ICE/PRO		---	---
User response	Declare this symbol as being PUBLIC in one of the source programs. Alternatively, modify the source program so that another symbol, declared as being PUBLIC, is referenced to.					

*

Error message (A406)	"Too many public symbol"					
Cause	The total number of public symbols exceeds the maximum limit, 65535.					
Program action	Aborts program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Reduce the total number of public symbols used in the source program to 65535 or less.					

Error message (A501)	"Insufficient memory in hostmachine"					
Cause	The system does not have sufficient memory to run the program.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Install additional memory in the host machine, if possible. Increase the amount of memory available to application programs. If additional memory cannot be installed, the program cannot be linked on this host machine.					

Error message (A901)	"Can't open device file '*filename*'"					
Cause	The device file cannot be opened.					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object	Load module	Without -JUNK specification	Link object	Load module
		---	---		---	---
User response	Check that the device file is in a valid directory.					

Error message (A904)	"Can't open output file '*filename*'"					
Cause	The output file cannot be opened (file I/O error).					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Check that the output file name is specified correctly. Check the status of the disk on which output file creation failed (free space, medium status, etc.).					

Error message (A905)	"Can't create temporary file '*filename*'"					
Cause	The temporary file for symbol entries cannot be created (file I/O error).					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Check the directory name specified with the -WORK option or the TMP environment variable. Check the status of the disk on which temporary file creation failed (free space, medium status, etc.).					

Error message (A907)	"Can't write output file '*filename*'"					
Cause	Data cannot be written to the output file (file I/O error).					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Check that the output file name is specified correctly. Check the status of the disk on which output file creation failed (free space, medium status, etc.).					

Error message (A908)	"Can't access temporary file '*filename*'"					
Cause	Data cannot be written to the temporary file (file I/O error).					
Program action	Aborts the program execution.					
File output	With -JUNK specification	Link object ---	Load module ---	Without -JUNK specification	Link object ---	Load module ---
User response	Check the directory name specified with the -WORK option or the TMP environment variable. Check the status of the disk on which temporary file creation failed (free space, medium status, etc.).					

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: 1-800-729-9288
1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-889-1689

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: 02-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Corporation
Semiconductor Solution Engineering Division
Technical Information Support Dept.
Fax: 044-548-7900

I would like to report the following error/make the following suggestion:

Document title: _____

Document number: _____ Page number: _____

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❑	❑	❑	❑
Technical Accuracy	❑	❑	❑	❑
Organization	❑	❑	❑	❑