H Series Linkage Editor, Librarian, and Object Converter

User's Manual

HITACHI

ADE-702-139
Rev. 1.0
12/18/96
Hitachi, Ltd.
McS-Setsu

Hitachi semiconductor

Notice

When using this document, keep the following in mind:

1. This document may, wholly or partially, be subject to change without notice.
2. All rights are reserved: No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without Hitachi's permission.
3. Hitachi will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's unit according to this document.
4. Circuitry and other examples described herein are meant merely to indicate the characteristics and performance of Hitachi's semiconductor products. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.
5. No license is granted by implication or otherwise under any patents or other rights of any third party or Hitachi, Ltd.
6. MEDICAL APPLICATIONS: Hitachi's products are not authorized for use in MEDICAL APPLICATIONS without the written consent of the appropriate officer of Hitachi's sales company. Such use includes, but is not limited to, use in life support systems. Buyers of Hitachi's products are requested to notify the relevant Hitachi sales offices when planning to use the products in MEDICAL APPLICATIONS.

Preface

This manual explains how to use the H Series Linkage Editor, Librarian, and Object Converter, which work on MS-DOS*[1] or UNIX*[2]. This manual consists of the following three parts:

Part I Linkage Editor Guide
Part II Librarian Guide
Part III Object Converter Guide

Users are encouraged to consult the user's manuals for other H Series cross-software. Relevant manuals include:

- H8S, H8/300 Series Cross Assembler User's Manual
- H8S, H8/300 Series C Compiler User's Manual
- H8S, H8/300 Series Simulator/Debugger User's Manual

- H8/500 Series Cross Assembler User's Manual
- H8/500 Series C Compiler User's Manual
- H8/500 Series Simulator/Debugger User's Manual

- SH Series Cross Assembler User's Manual
- SH Series C Compiler User's Manual
- SH Series Simulator/Debugger user's Manual

Notes: 1. MS-DOS is an operating system administrated by Microsoft Corporation.
2. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Notes:

The following symbols have special meaning in this manual.

<item>	:	Specification item
{ }	:	One of the items between the brackets is to be selected.
[ ]	:	The enclosed item is optional (i.e., can be omitted)
...	:	The preceding item can be repeated.
Δ	:	Blank space(s) or tab(s)
(RET)	:	Press the Return (Enter) key.

File extensions are in uppercase letters on MS-DOS.

Hexadecimal data in this manual is prefixed by H'. (Example: H'1000)

Data without prefix is in decimal unless otherwise specified.

Contents

Part I Linkage Editor Guide

Figures

Part I

Part II

Part III

Tables

Part I

Part II

Part III

Part I

Linkage Editor Guide

Section 1 Overview

The growing need for large-scale, complex microcomputer programs has led to the common practice of developing a program in separate parts and using a high-level language. In generating a program in this fashion, a compiler or an assembler is used to convert source programs into object modules. After that, a linkage editor is employed to link and edit the modules into one load module file.

The H Series Linkage Editor (hereafter, referred to as the Linkage Editor) inputs object module files output by an assembler or C compiler, links and edits them, and generates a single load module file.

Figure 1-1 illustrates the program development procedure using the Linkage Editor.

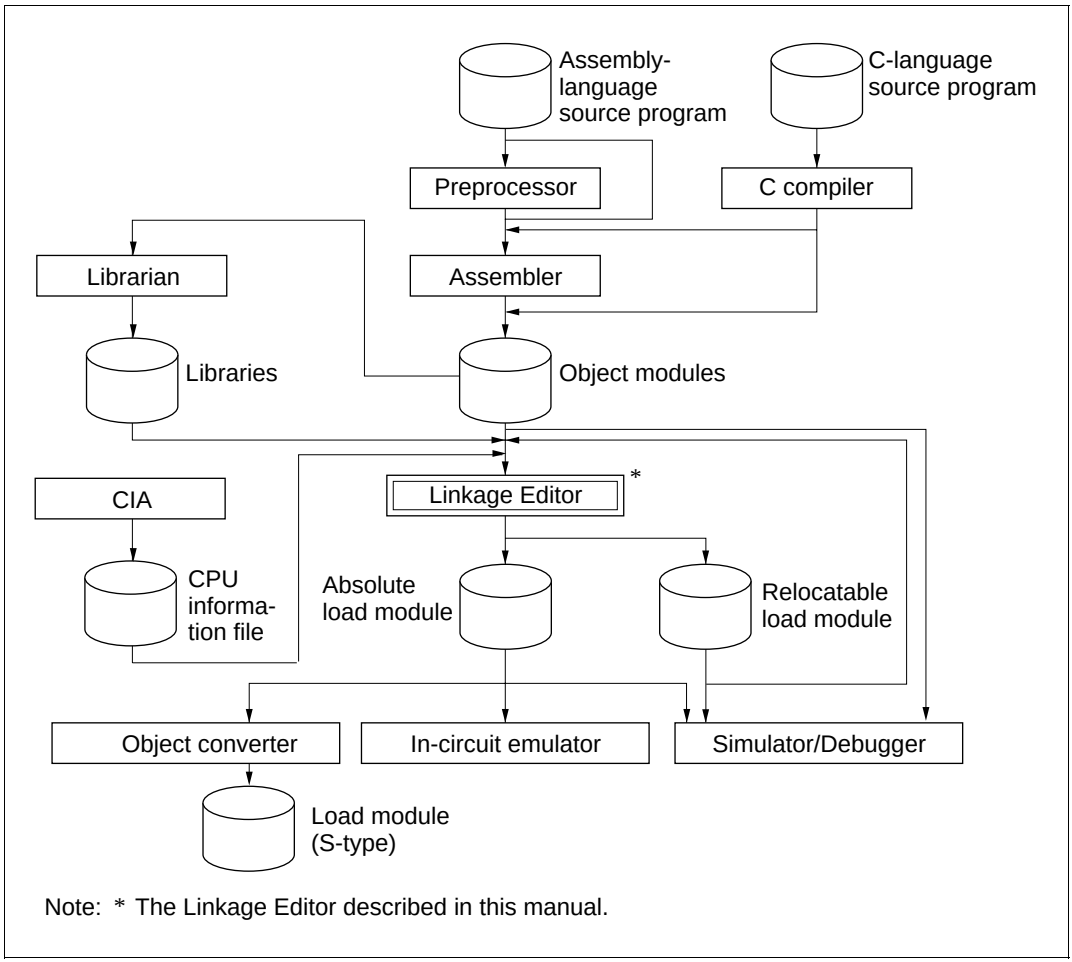


Figure 1-1 Program Development Procedure

The Linkage Editor has the following features:

(1) Linkage can be executed by command-line specifications or by subcommands. These two methods allow flexible control over the Linkage Editor to match the desired application.

(2) The load module file output by the Linkage Editor can be re-input and re-edited to generate a new load module file.

(3) Data used by a simulator/debugger or in-circuit emulator in symbolic debugging can be included in the load module file by specifying options.

1.1 Linkage Editor Functions

The Linkage Editor provides the following five basic functions.

Module Linkage: The module linkage function links and edits object modules output by a compiler or assembler.

Address Resolution: The address resolution function determines absolute addresses for external reference symbols so that references can be made between modules. It also determines absolute addresses for relative addresses.

Load Module File Re-input: The re-input function enables a load module file output by the Linkage Editor to be input again.

Multilinkage: The multilinkage function enables the linkage process to be carried out multiple times during one execution of the Linkage Editor.

Debugging Support: The debugging support function allows display of interim linkage results and provisional correction of errors.

1.2 Object Module and Load Module

An object module is output as a result of compiling or assembling a source program. A load module is obtained by using the Linkage Editor to link object modules.

There are two load module formats: absolute and relocatable. An absolute load module has been assigned absolute addresses, and is in executable form. It does not contain relocation information for relinking and relocation. A relocatable load module has been assigned relative addresses and contains relocation information. This information enables the relocatable load module to be re-input into the Linkage Editor for relinking and relocation. The load module format is selected by the FORM option or subcommand. For details on the FORM option and subcommand, refer to section 4.5.3, “FORM—Specifies Output Load Module File Format.”

Object modules, absolute load modules, and relocatable load modules are collectively referred to as modules in this manual.

Modules are either page type or non-page type, depending on the H series microcomputer. The two types differ as to the method of assigning addresses when modules are linked. H8/500 series modules are page type, whereas H8S, H8/300 series and SH series modules are non-page type.

1.3 Unit and Section

A unit in a module refers to a compile unit or assembly unit. An object module output by a compiler or assembler consists of a single unit. A load module which represents multiple object modules that have been linked by the Linkage Editor contains more than one unit.

A unit is divided into sections. The Linkage Editor processes one section at a time.

The interrelation among module, unit, and section is illustrated in figure 1-2.

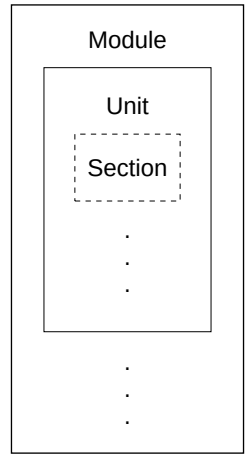


Figure 1-2 Interrelation among Module, Unit, and Section

A section has a name for identification, an attribute describing its content and usage, and a format: either absolute or relocatable. Even if two sections have the same name, they are treated as separate sections when their attributes or formats are different.

Section attributes and formats are classified as follows.

(1) Attributes

- Code: An area containing instructions or constants.
- Data: A variable area with values that are changed by the program.
- Stack: A stack or work area which cannot be initialized.
- Common: A variable area used in common by multiple modules.
- Dummy: Used, for example, to define the structure of a variable area; does not generate any actual object code.

(2) Formats

- Absolute: A section in which absolute addresses have already been assigned.
- Relocatable: A section in which absolute addresses have not yet been assigned.

Section 2 Linkage Editor Functions

This section gives a more detailed description of the basic functions provided by the Linkage Editor. The following discussion and examples will make reference to various options and subcommands used to control the Linkage Editor. Additional information on these options and subcommands can be found in section 3, “Executing the Linkage Editor,” and section 4, “Linkage Editor Options and Subcommands.”

2.1 Module Linkage

The Linkage Editor reads modules from specified input files and links these modules to generate one load module. Modules are linked by each section, a section being the smallest complete part making up a module.

2.1.1 Section Linkage

A section is linked only if it is relocatable. Since absolute sections have already been assigned absolute addresses, no further linking is performed. Relocatable sections are linked according to the procedure described below.

(1) Grouping of Sections with the Same Name

Sections having the same name but found in more than one unit are grouped.

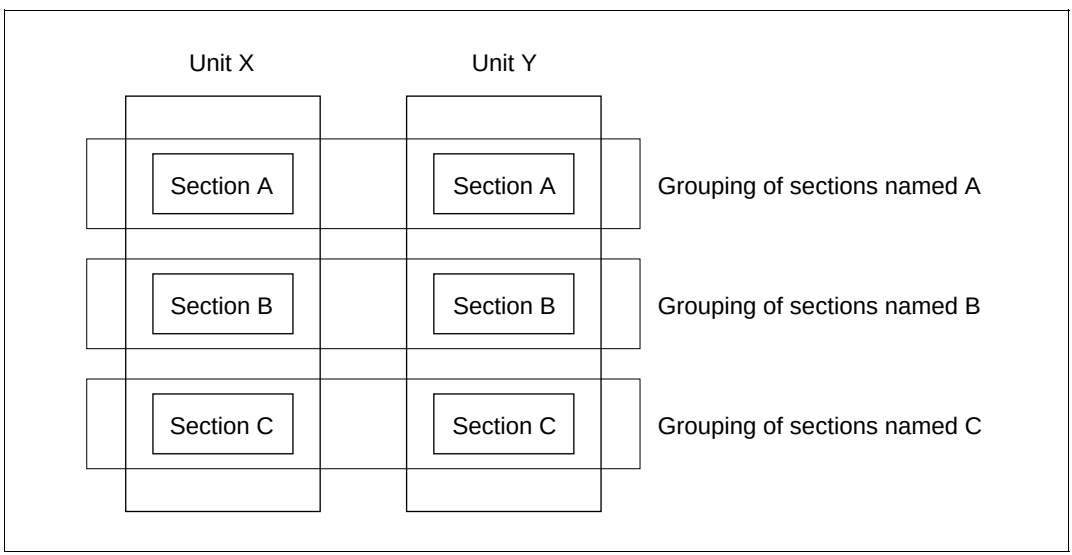


Figure 2-1 Grouping Sections Having the Same Name

A warning message is output when sections have the same name but different attributes. Such sections are then processed as separate sections.

(2) Linking of Sections with the Same Name

Sections having the same name are linked in one of three ways, depending on their attributes.

(a) Simple linkage

Sections with the code, data, or stack attribute and having the same name are allocated consecutively, in the order in which the modules were input.

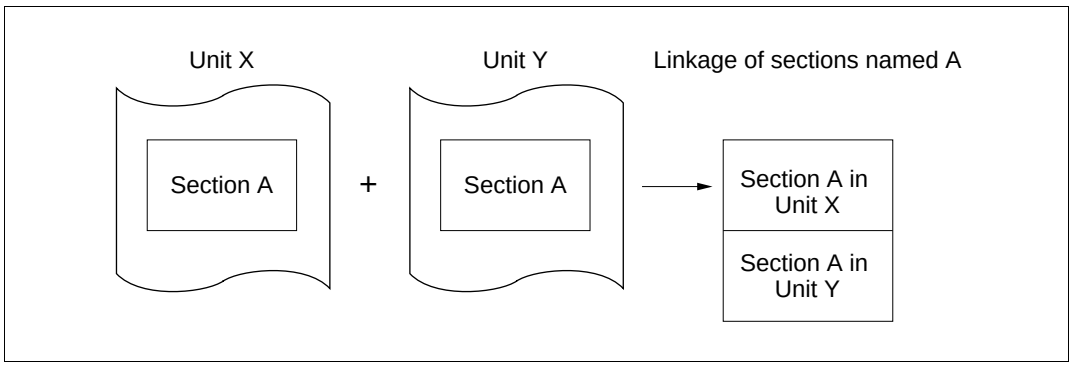


Figure 2-2 Simple Linkage

(b) Common linkage

Sections with the common attribute and having the same name are allocated at the same address. The address area allocated is equal to the size of the largest section.

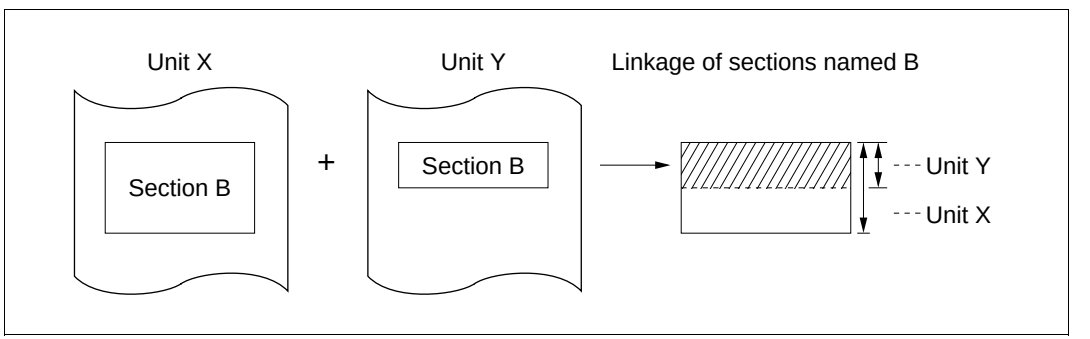


Figure 2-3 Common Linkage

(c) Dummy linkage

Sections with the dummy attribute are not linked, because they do not have any actual existence in the object module file.

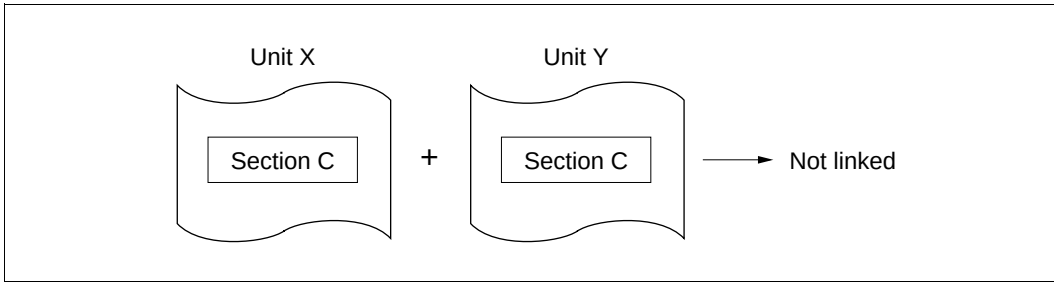


Figure 2-4 Dummy Linkage

(3) Linking of different sections

If a section linking order is specified when the Linkage Editor is executed, sections are linked in that order. If the section linking order is not specified, sections are linked in the order in which they were input.

(a) With a specified linkage order

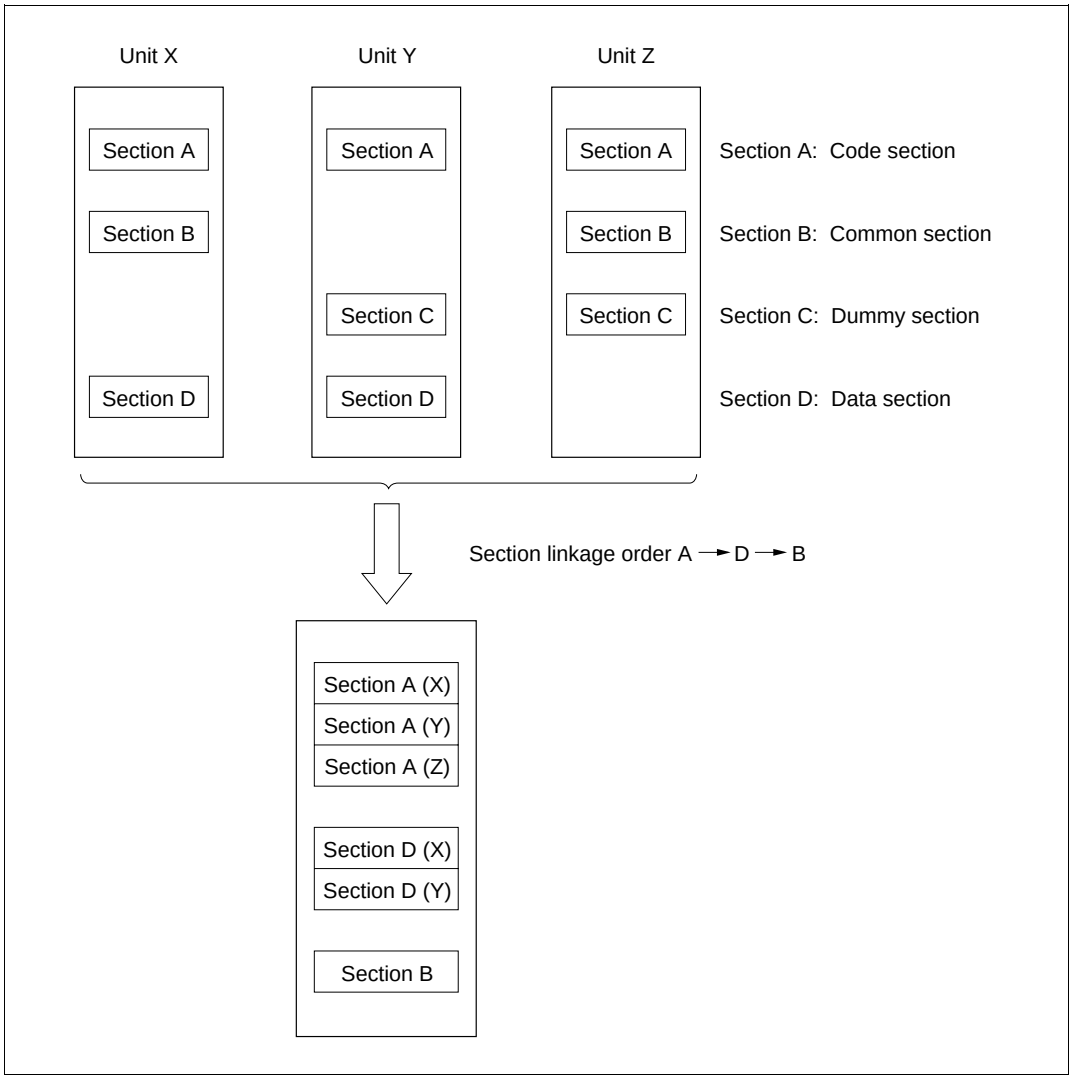


Figure 2-5 Example of Section Linkage with a Specified Linkage Order

The section linkage order can be specified only when the load module output by the Linkage Editor has the absolute format. The linkage order is specified using the START option or subcommand.

(b) Without a specified linkage order

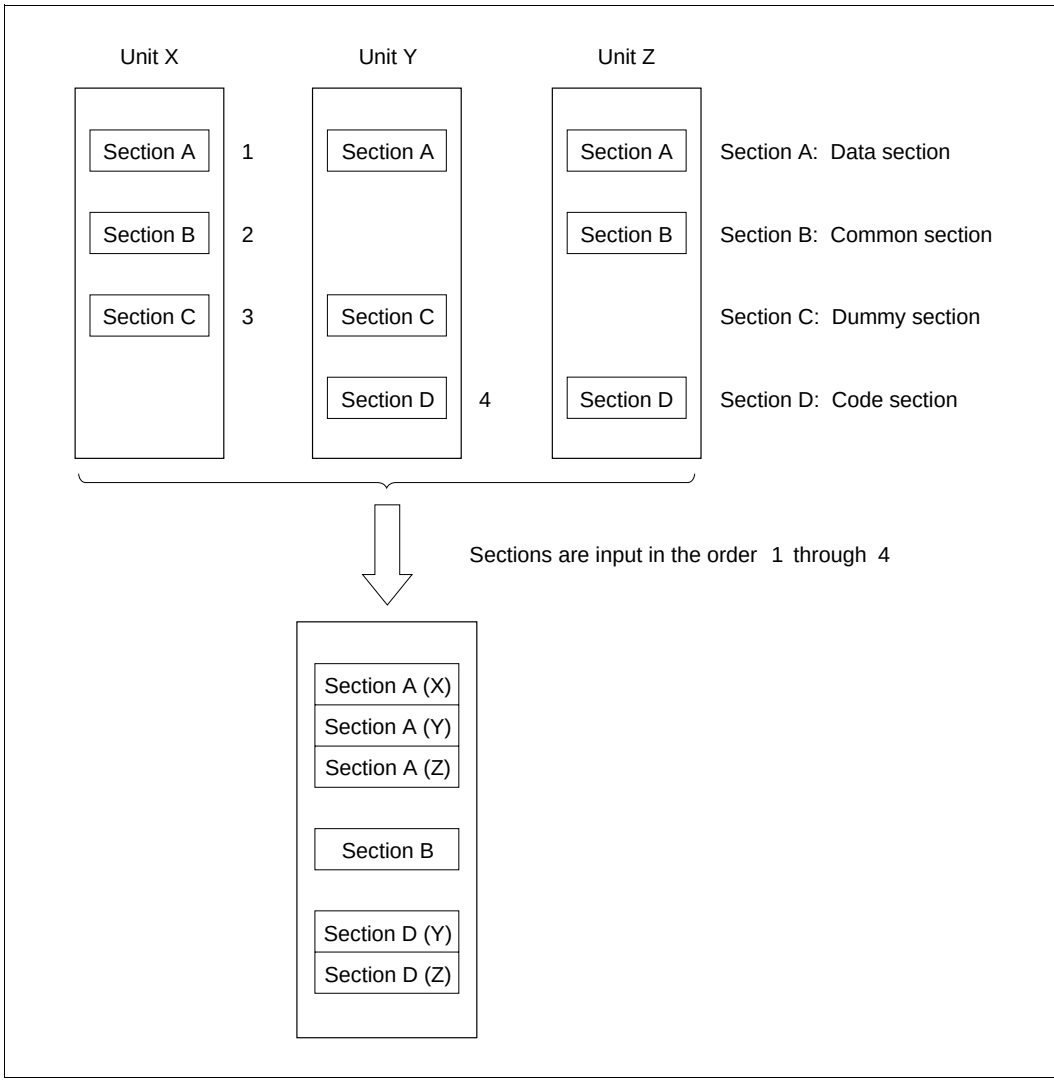


Figure 2-6 Example of Section Linkage without a Specified Linkage Order

Sections having the same name but different attributes are linked in the order in which they are input.

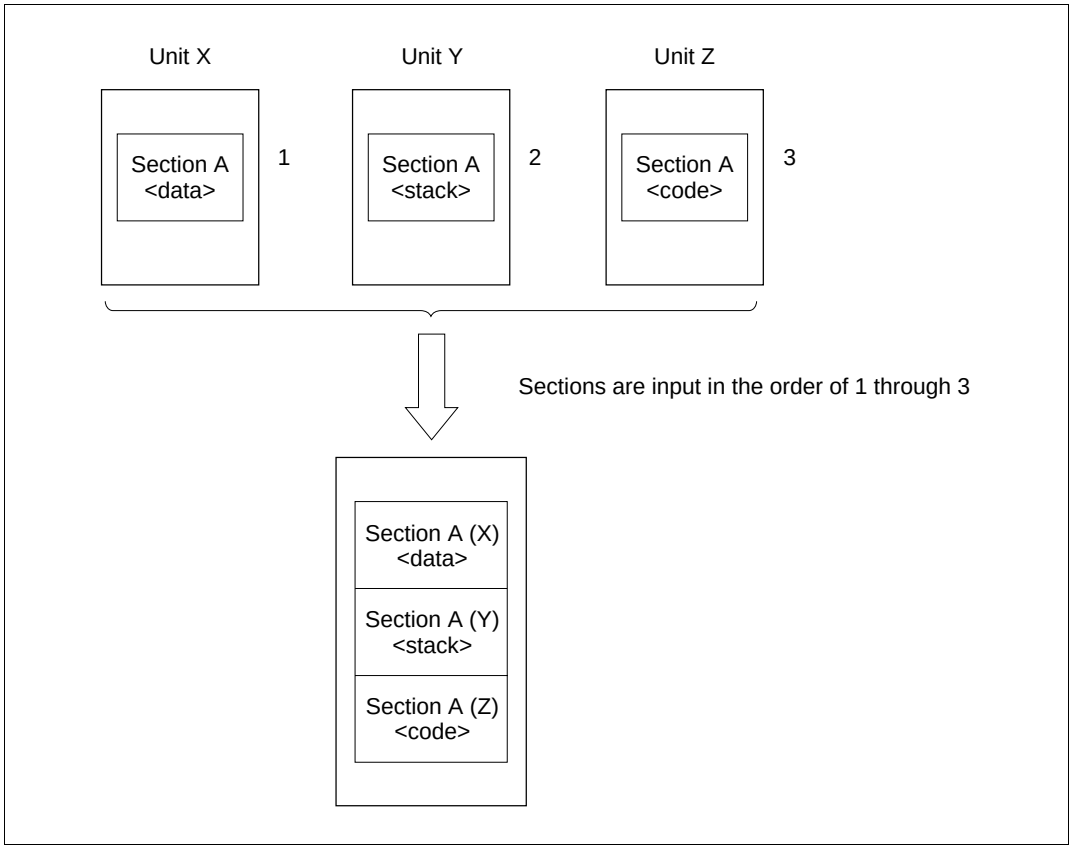


Figure 2-7 Example of Section Linkage for Same Section Name but Different Attributes

(4) Address assignment

Addresses are assigned to each section. Absolute addresses are assigned when the output load module file has the absolute format. The section linkage order and start address can be specified using the START option or subcommand. Absolute addresses are assigned to each section in order, beginning with the start address. If no start address is specified, absolute addresses are assigned beginning from address zero.

If sections with absolute format are linked to sections with relocatable format, the same absolute address may be assigned to more than one section. In that case, the Linkage Editor displays a warning message.

When page type modules are linked, if addresses are assigned section by section, one section may overlap a page boundary. In this case the Linkage Editor will display a warning message. However, executing a load module one of whose sections overlaps a page boundary is extremely troublesome. For this reason the Linkage Editor is provided with an autopaging function, which prevents any section in a unit from overlapping the page boundary by allocating the section to the top of the next page. Use of this function is designated by means of the AUTOPAGE option or subcommand. The different methods of assigning addresses to page type modules are shown in figure 2-8 (neither autopaging nor start address specified), figure 2-9 (autopaging specified, start address not specified), and figure 2-10 (autopaging and start address specified).

When the output load module file has the relocatable format, addresses in each section are assigned relative to the beginning of the section. The output format is specified using the FORM option or subcommand.

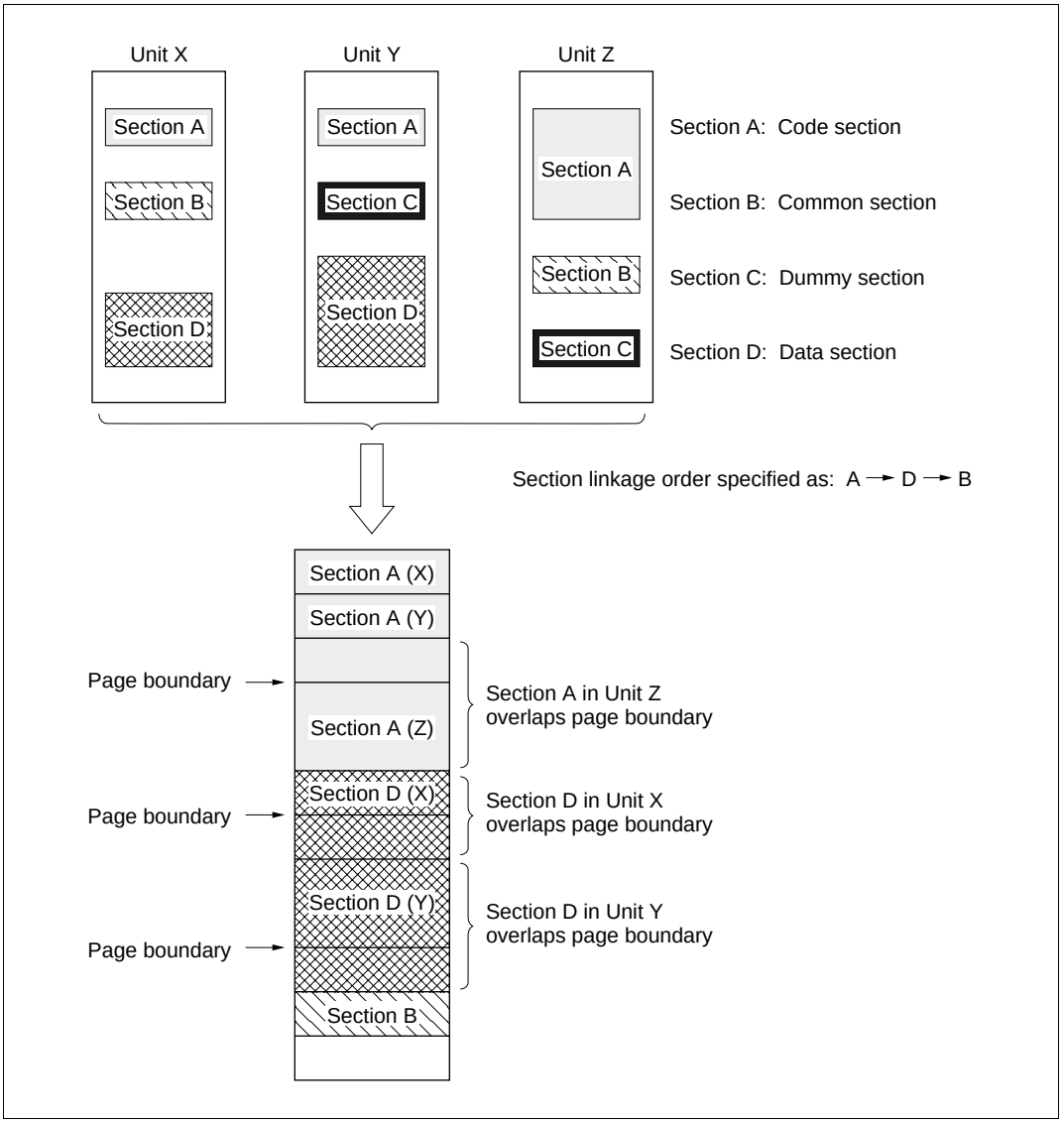


Figure 2-8 Linking of Page Type Modules (Neither Autopaging nor Start Address Specified)

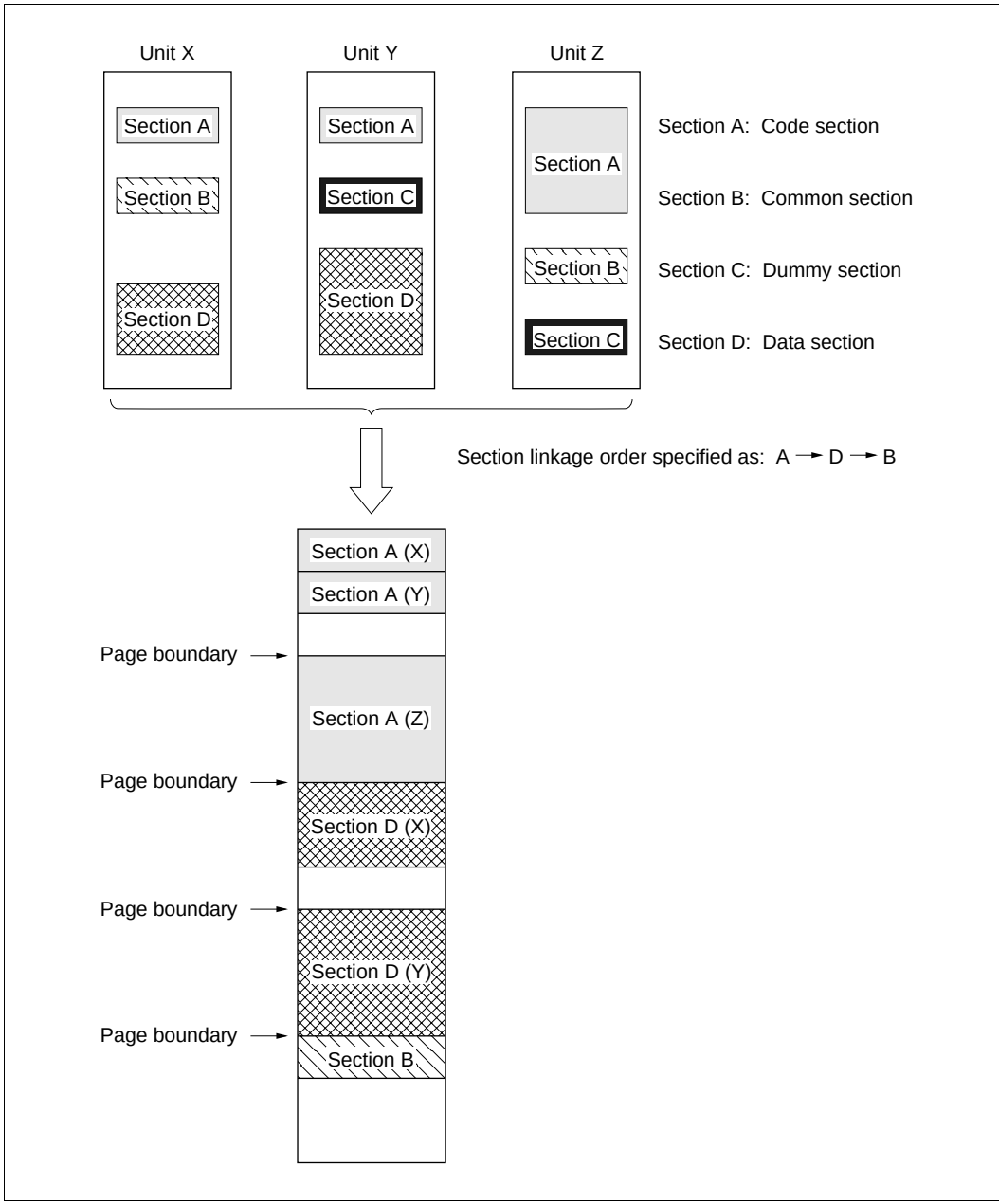


Figure 2-9 Linking of Page Type Modules (Autopaging Specified, Start Address Not Specified)

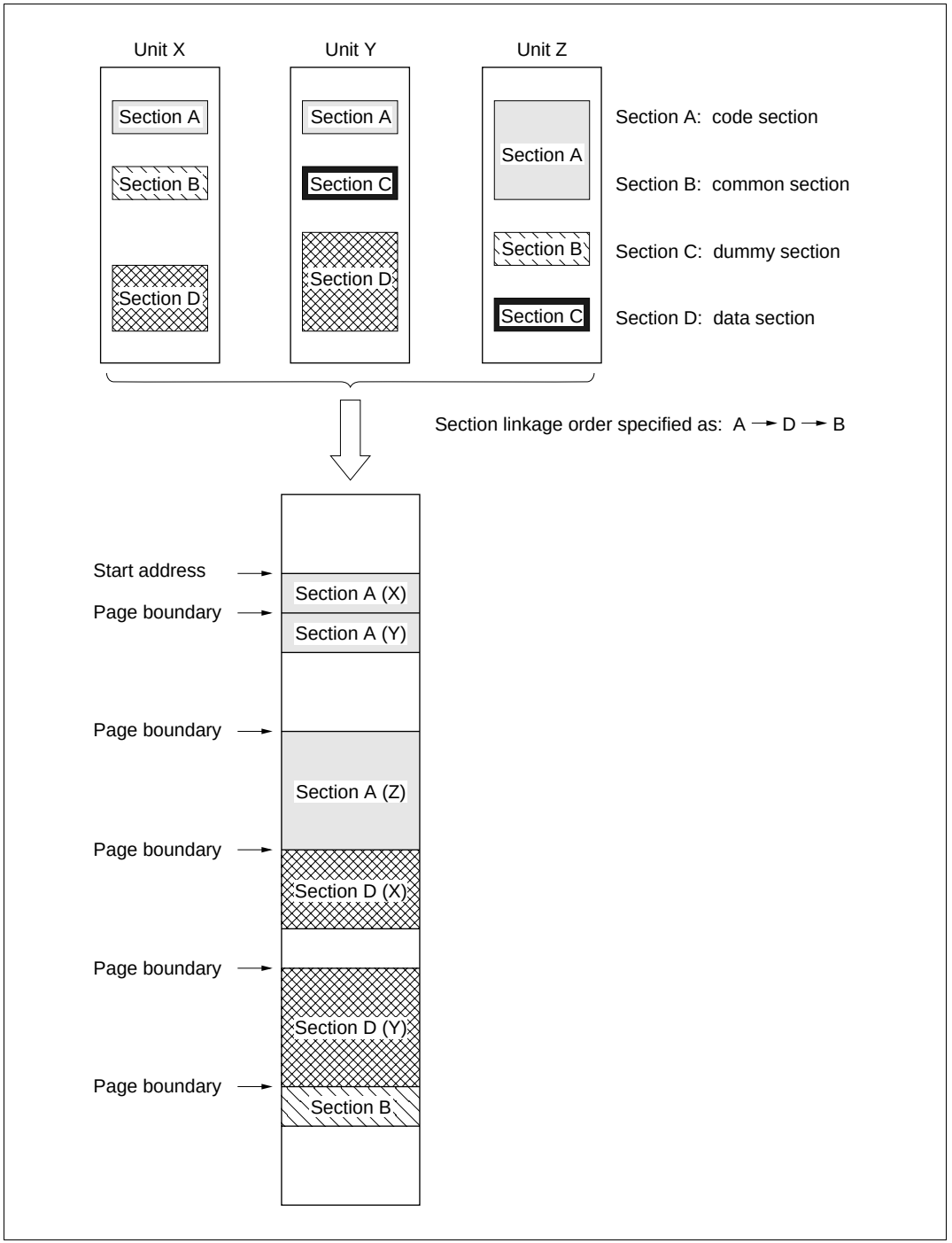


Figure 2-10 Linking of Page Type Modules (Autopaging and Start Address Specified)

2.1.2 Inclusion from Library Files

The Linkage Editor can link object modules and relocatable load modules input from library files created with the H Series Librarian, and include these modules in the output load module. Inclusion from library files is accomplished in either of the following two ways.

(1) Inclusion by Specifying the Module Name: Particular modules in a library file can be included by specifying the library file name and module name when input file names are specified. Input file names are specified on the command line or by the INPUT subcommand.

(2) Automatic Inclusion: After all specified modules have been input, the Linkage Editor begins resolving external reference symbols (after this, external reference symbol is called “import symbol”). If an import symbol is not defined in any of the modules, the Linkage Editor searches the specified library files. If it finds a module defining the unresolved import symbol, the Linkage Editor automatically inputs and links this module. If the unresolved import symbol is not defined in any of these library files, the Linkage Editor searches one or more default library files defined in advance by the user. Again, if it finds a module defining the unresolved import symbol, the Linkage Editor automatically inputs and links this module.

If no module in the default libraries defines the unresolved import symbol, an undefined import symbol error occurs.

A detailed explanation of default libraries is given in section 5.4, “Default Library Files.”

Library files are classified into system library files and user library files. The Linkage Editor first searches user library files. When modules containing externally defined symbols (after this, externally defined symbol is called “export symbol”) of the same name exist both in a specified system library file and in a user library file, the module in the user library file is linked. The order in which two or more user library files or system library files are searched depends on the order in which they are specified.

A library file can contain both page type and non-page type modules. If both types of modules are input into the Linkage Editor at the same time, an error will occur. Care must therefore be taken both when creating library files and when specifying them.

Library files are specified using the LIBRARY option or subcommand. On the designation of library files as system files or user files, see Part II, Librarian Guide.

An example of the order of module linking when library files are specified is given below.

(1) Object modules a and b are input by the INPUT subcommand.

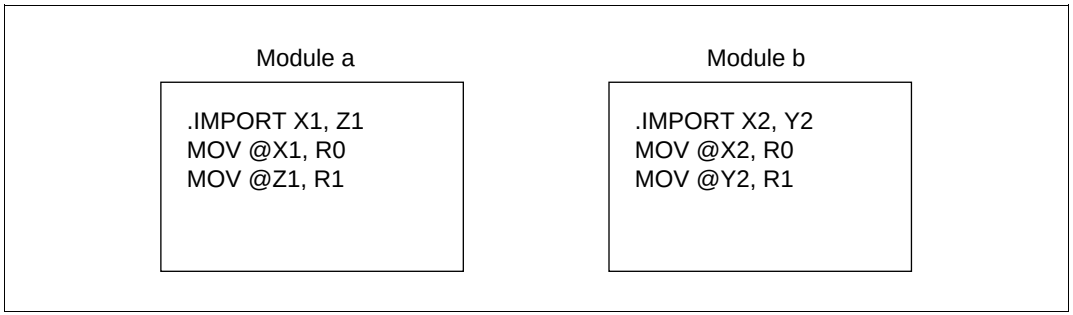


Figure 2-11 Example of Module Linking (Input Object Modules)

(2) Library files lib1, lib2, and lib3 are input in that order by the LIBRARY subcommand.

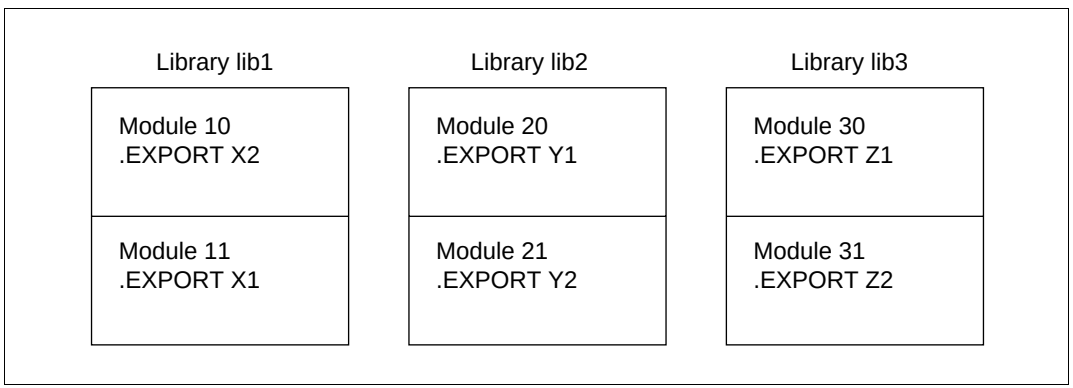


Figure 2-12 Example of Module Linking (Input Library Files)

(3) The Linkage Editor first collects all import symbols declared in the input files, then searches for export symbols in the first specified library. If a symbol is found, the module defining it is linked.

If two or more symbols are declared in separate modules in the same library, the modules are linked in their order of appearance in the library. If a symbol is not found in that library, the next specified library is searched.

In the above example, modules are linked in the following order.

Module a
Module b
Module 10
Module 11
Module 21
Module 30

Figure 2-13 Example of Module Linking (Output Load Module)

2.1.3 Exclusion of Module Linking

An option or subcommand selects whether or not to link modules that define non-referenced import symbols. In the following coding example symbol abc is declared as an import symbol, but is not referenced in any executable statement. If exclusion is specified, the module defining symbol abc in a library file will not be linked.

```
.IMPORT xyz, abc
MOV.W @xyz, R0
.
.
.
.END
```

Figure 2-14 Example of Module Containing Non-Referenced Import Symbol

In a C language program, import symbols are described by an extern declaration, but these symbols are not necessarily referenced. (For example, a large number of non-referenced import symbols are declared in stdio.h.) The exclusion function reduces program size by excluding unnecessary modules. Exclusion of such modules is specified by the EXCLUDE option or subcommand.

2.2 Address Resolution

When a source program is assembled, the absolute addresses of certain symbols cannot be decided. These include symbols imported from another module and symbols in relocatable sections of the same module. The Linkage Editor determines absolute addresses for these symbols and sets the absolute addresses to the reference positions.

2.2.1 Import Symbol Resolution

When importing symbols from a separate module, the assembler outputs import information in the object program. It also declares export of symbols that can be imported in other modules. As a result, export information is output in the object program. The Linkage Editor relates this import and export information. In addition, it uses address information specified by options or subcommands to determine absolute addresses for the export symbols, and replaces corresponding import symbols with the absolute addresses.

The example given in figure 2-15 illustrates how import symbols are resolved. The modules, sections, and subcommands used in the figure are explained below.

(1) Module a

- This module consists of one section, section X, having a size of 5000 (hexadecimal) bytes.
- Symbol S4 in module b is imported at position A1.
- Symbol S2 in module b is imported at position A2.

(2) Module b

- This module consists of sections X and Y.
- The size of section X is 2000 (hexadecimal) bytes.
- The size of section Y is 3000 (hexadecimal) bytes.
- S1 is the start of section Y. S2 is located 1000 (hexadecimal) bytes from S1.
- S3 is the start of section X. S4 is located 1200 (hexadecimal) bytes from S3.

(3) Module c

- This module consists of one section, section Z, having a size of 4000 (hexadecimal) bytes.
- Symbol S3 in module b is imported at position C1.
- Symbol S1 in module b is imported at position C2.

(4) Subcommands

```
INPUT∆a, b, c
START∆X, Y, Z(10000)
EXIT
```

Three modules a, b, and c are input to the Linkage Editor. Sections are linked in the order X, Y, Z. The start address is 10000 (hexadecimal).

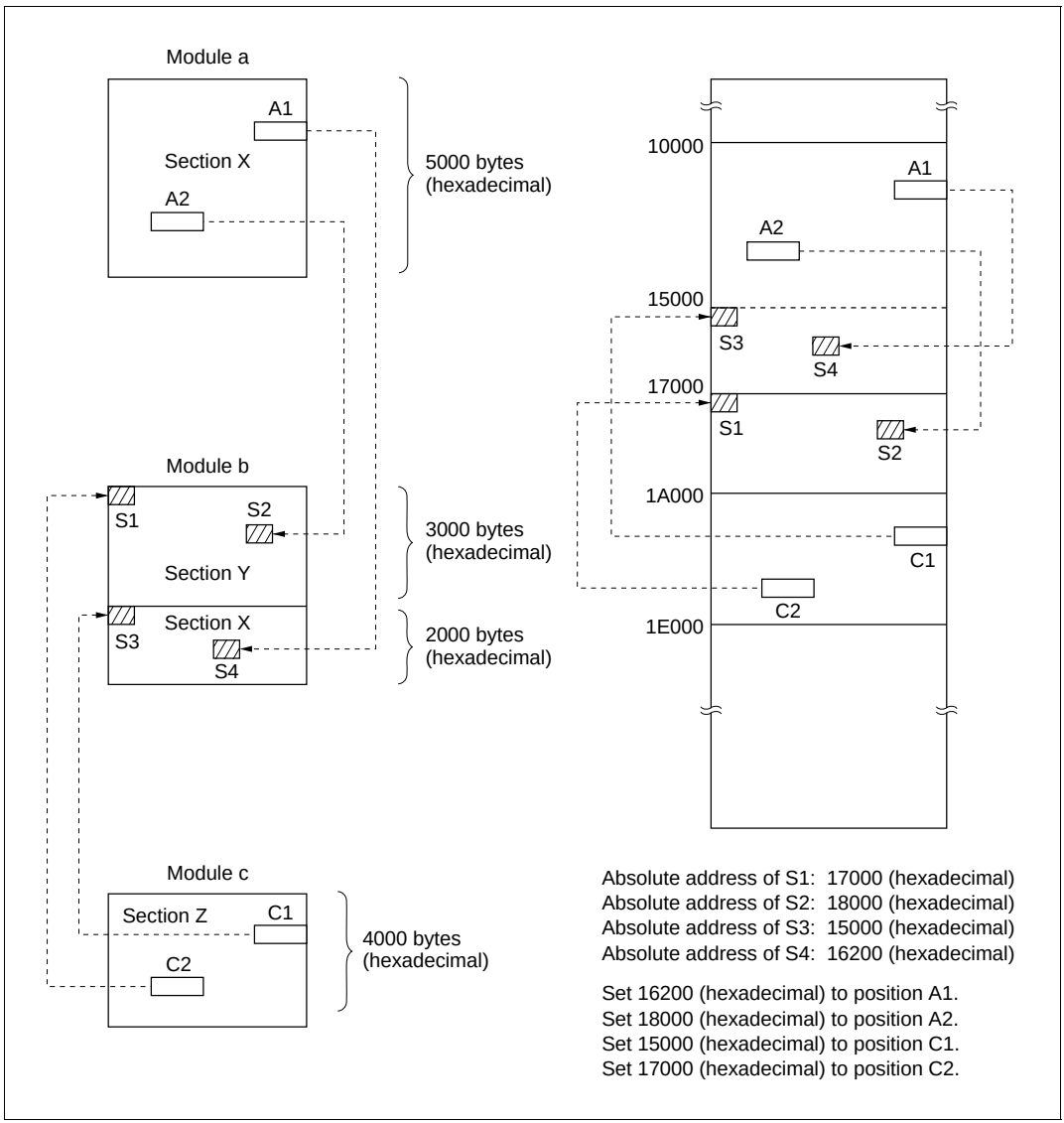


Figure 2-15 Resolution of Import Symbols

2.2.2 Address Resolution within a Module

When a symbol defined in a relocatable section of a module is referenced within the same module, the assembler expresses the symbol address as a relative address from the start of the section. The Linkage Editor uses this relative address value and address information specified by options or subcommands to decide the absolute address. It then replaces the relative address with the absolute addresses.

The example given in figure 2-16 illustrates the resolution of addresses within a module. The modules, sections, and subcommands used in the figure are explained below.

(1) Module a

- This module consists of one section, section X, having a size of 5000 (hexadecimal) bytes.

(2) Module b

- This module consists of sections X, Y, and Z.
- The size of the section X is 6000 (hexadecimal) bytes.
- The size of the section Y is 1000 (hexadecimal) bytes.
- The size of the section Z is 2000 (hexadecimal) bytes.
- B1 references S1.
- B2 references S3.
- B3 references S2.
- S1 is located 3000 (hexadecimal) bytes from the start of section X.
- S2 is located 4500 (hexadecimal) bytes from the start of section X.
- S3 is located 5000 (hexadecimal) bytes from the start of section X.

(3) Subcommands

```
INPUT∆a, b
START∆X, Y, Z(10000)
EXIT
```

Two modules a and b are input to the Linkage Editor. Sections are linked in the order X, Y, Z. The start address is 10000 (hexadecimal).

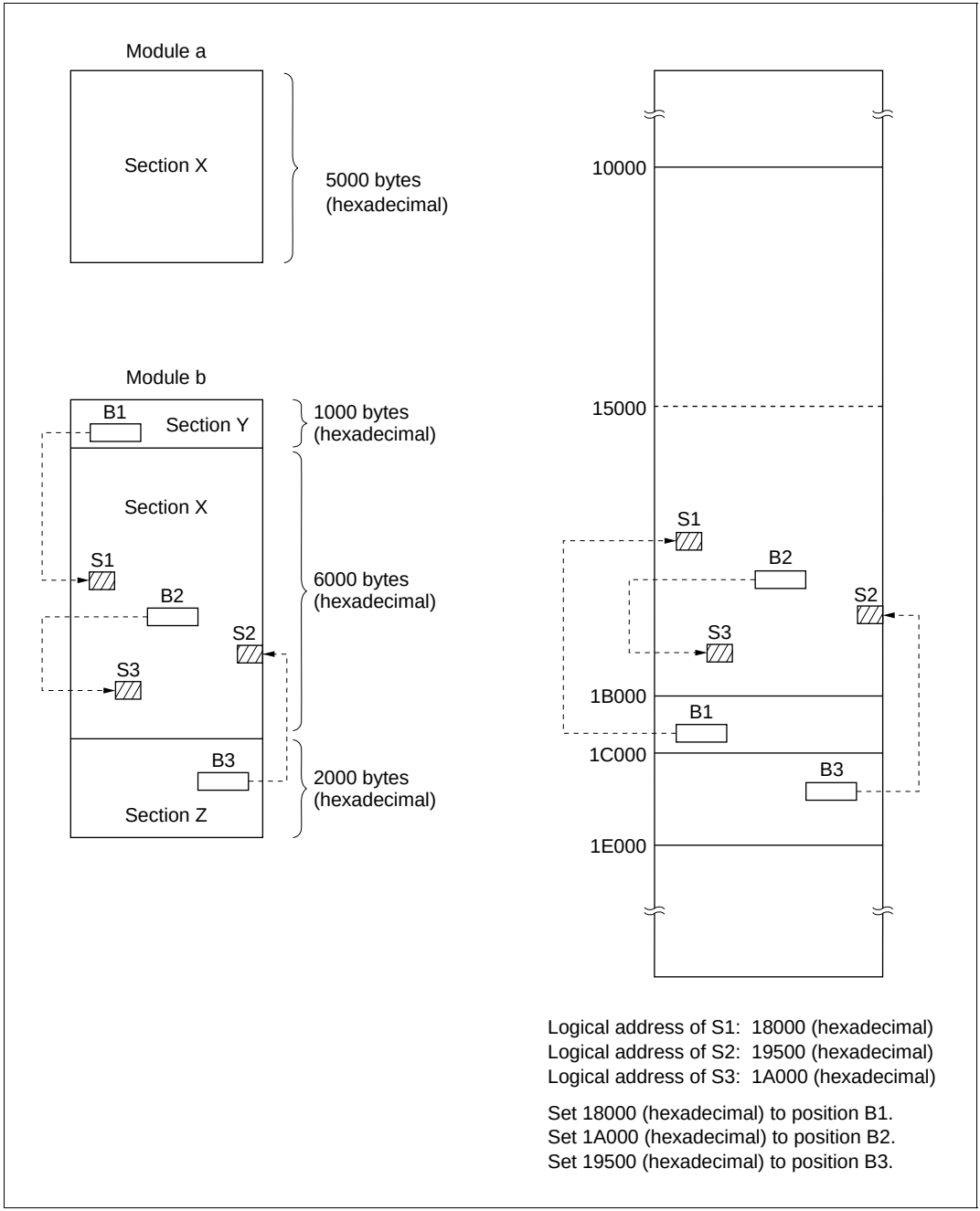


Figure 2-16 Address Resolution within a Module

2.2.3 Suppressing the Listing of Unresolved Symbols

For a relocatable load module, the display of unresolved symbol names can be suppressed. This can be selected by the UDF option or subcommand.

2.3 Load Module File Re-Input

Load module files have to be recreated using the Linkage Editor when a program has been modified or import symbols remain unresolved. The re-input function eliminates the need to specify each object module separately. By simply specifying the existing load module file and the object module files that were modified (or the object module files containing the export symbols), this function will recreate the load module file.

If modules are to be replaced, the re-input function carries out the replacement on a unit basis. A detailed explanation of unit replacement is given in section 2.3.1, “Automatic Unit Exchange.”

The load module file to be re-input can be specified on the command line or using the INPUT subcommand.

Only load module files in relocatable format can be re-input. The FORM option or subcommand is used to specify the relocatable format when creating a load module file.

An overview of the load module file re-input function is shown in figure 2-17.

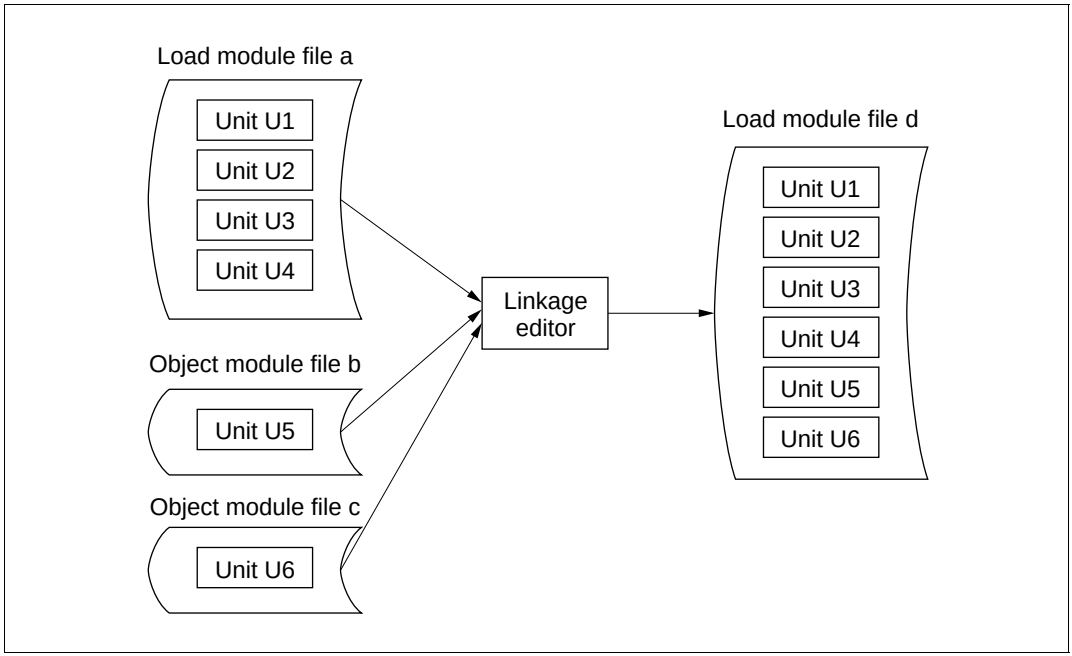


Figure 2-17 Load Module File Re-Input Function

Load module file a and object module files b and c are input to the Linkage Editor, which outputs a new load module file d. Load module file d consists of units U1, U2, U3, U4, U5, and U6.

2.3.1 Automatic Unit Exchange

When the Linkage Editor finds units with the same name in two or more modules, it gives inclusion priority to the unit in the module that was specified first. To replace units in a load module file, first specify files containing the replacement units, then specify the relevant load module file. This will produce the same result as using the EXCHANGE subcommand. This function is called automatic unit exchange.

By using automatic unit exchange, new load module files can be created by simply changing the specified order of file input. This feature is convenient when it is necessary to modify programs frequently, such as during debugging.

An example of the procedure for automatic unit exchange is shown in figure 2-18.

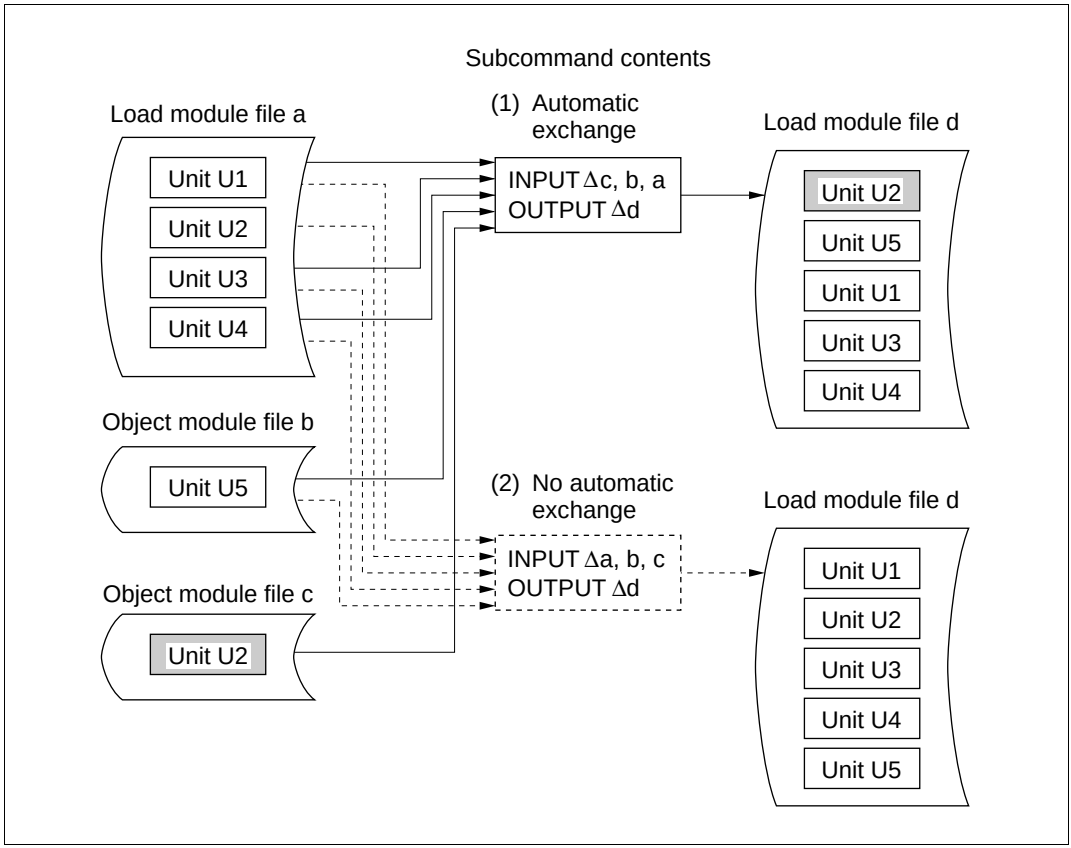


Figure 2-18 Automatic Unit Exchange

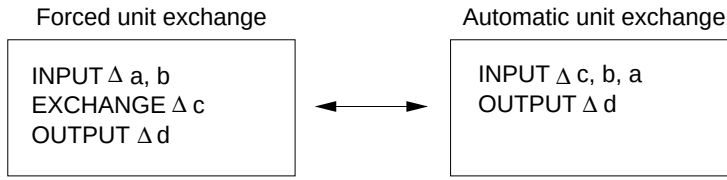
(1) Automatic Exchange: Object module files c and b and load module file a are input in that order. Unit U2 in load module file a is not included by the Linkage Editor since unit U2 in load module file c has already been input.

(2) No Automatic Exchange: Load module file a and object module files b and c are input in that order. Unit U2 in load module file c is not included by the Linkage Editor since unit U2 in load module file a has already been input.

2.3.2 Forced Unit Exchange

In addition to using automatic unit exchange, the EXCHANGE subcommand can also specify the units to be replaced. This function is called forced unit exchange.

By specifying the following subcommands, the result of forced unit exchange will be the same as that of the automatic unit exchange shown in figure 2-18.



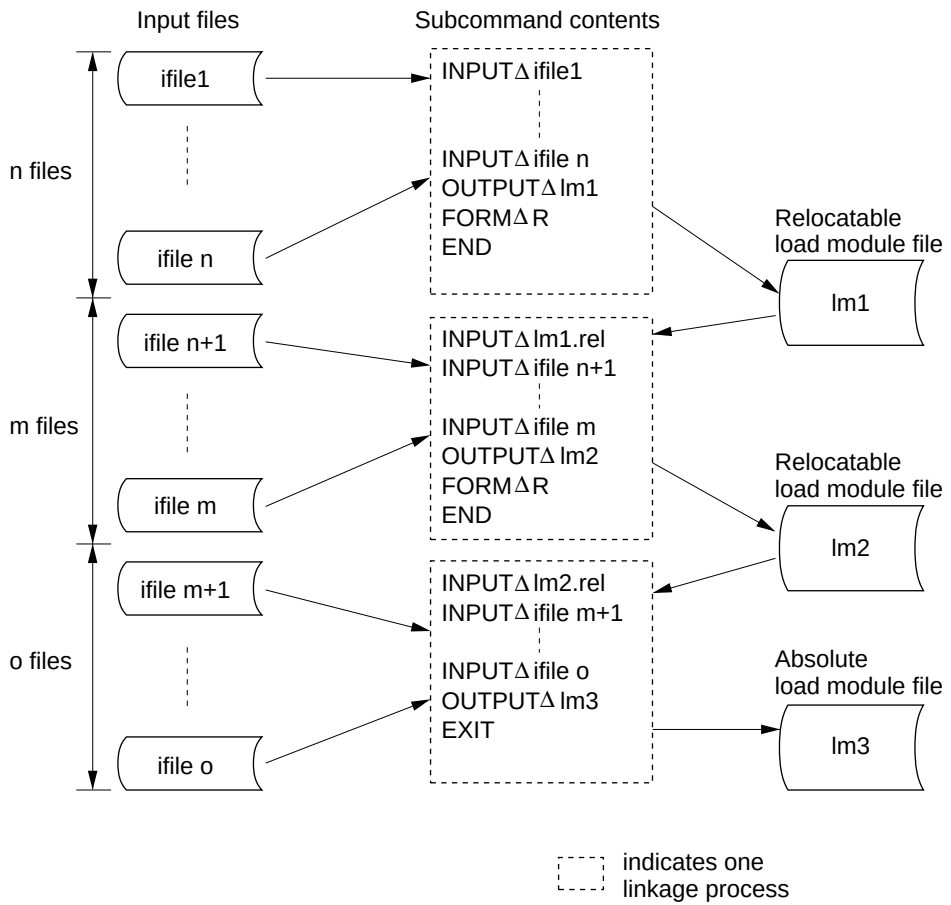
In this example of forced unit exchange, the Linkage Editor inputs units U1, U2, U3, and U4 in load module file a and unit U5 in object module file b, then forcibly replaces the unit U2 already input with unit U2 in object module file c. Load module file d output by the Linkage Editor contains units U1, U3, and U4 from file a, unit U5 from file b, and unit U2 from file c. Thus load module file d has the same unit configuration as load module file d shown in the example of automatic file exchange in figure 2-18.

2.4 Multilinkage

The Linkage Editor can handle up to 256 input files in one linkage process. When there are multiple input files, one way to link them is to re-input the load module file. The multilinkage function allows several linkage processes to be completed with just one execution of the Linkage Editor, instead of executing it separately for each linkage process.

The END subcommand indicates the end of one linkage process of the multilinkage function. The end of the final linkage process, however, is specified by the EXIT subcommand.

An example of the multilinkage function is shown in figure 2-19.



Note: When the default library is used during multi-linkage process, the modules in the default library are linked in the first linkage process. When the modules must be linked in the final linkage process, specify the NOLIBRARY command in the processes except the final process.

Figure 2-19 Multilinkage Function

2.5 Debugging Support

Debugging support functions confirm the interim linkage results at the program debugging stage and make provisional recovery from errors in load module files. Debugging support functions include displaying interim linkage information as well as defining, changing, and deleting export and import symbol names. A brief explanation of each function is given below.

(1) Display of Interim Linkage Information: This function is used during subcommand input when it is desired to see information about the load module being processed by the Linkage Editor. Specifying the LIST subcommand outputs interim linkage information to the standard output device.

Three types of linkage information are displayed.

(a) Linkage map
(b) Unresolved import symbols
(c) Export symbols

(2) Change and Deletion of Unit Names, Export Symbol Names, and Import Symbol Names: These functions can change or delete any duplicated names of units, export symbols, and import symbols. Noted that names of import symbols cannot be deleted.

Names are changed by the RENAME subcommand and are deleted by the DELETE subcommand.

(3) Forced Definition of Import Symbols: This function defines provisional values for import symbols. The values defined with this function are valid only for the linkage operation being processed.

The forced definition of these symbol values is specified using the DEFINE option or subcommand.

2.6 Address Check

When an absolute load module is created with the Linkage Editor, addresses must be assigned to sections in accordance with the target CPU memory map. If not, the load module cannot be loaded to memory.

The address check function provided with the Linkage Editor confirms the validity of section address assignments on the basis of CPU memory map information (hereinafter called “CPU information”). This CPU information is read from a specified file.

To check an address, the CPU option or subcommand specifies the CPU information file. The CPU information file is created using the CPU information analysis program (CIA) included in the simulator/debugger. Note that the CPU information analysis program is not available for CPUs other than the H8S, H8/300, and SH series; thus the address check function can be used only with these series.

Regarding the method of creating a CPU information file, refer to the H8/300 Series or SH Series Simulator/Debugger User’s Manual or the SH Series Simulator/Debugger User’s Manual.

2.7 Support of Storing Program in ROM

When a user program is coded in C language and the load module is to be stored in ROM, data sections having initial value (D sections) will also be stored in ROM. To assist the user, the Linkage Editor carry out the following operations.

(1) An area of the same size as the D section (called the D' section) is reserved in the RAM area of the output load module. The memory map of the load module looks like this:

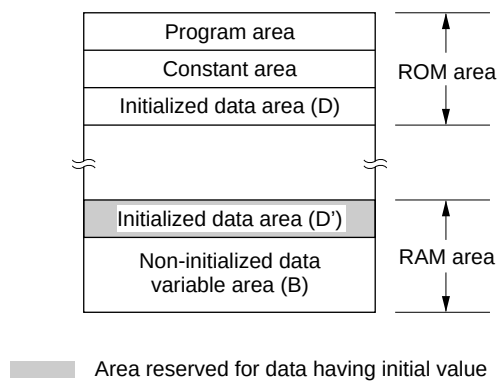


Figure 2-20 Memory Map for Storing Program in ROM

(2) When a variable declared in the D section is referenced, its address is changed to point to the RAM area. The variable address becomes:

Start address of D section + relative address within section

The ROM ability support function changes this to:

Start address of D’ section + relative address within section

Example: MOV @a, R0

The address of symbol “a” declared in the D section becomes (x) + (y) as shown in figure 2-21. This address is also stored on the object code.

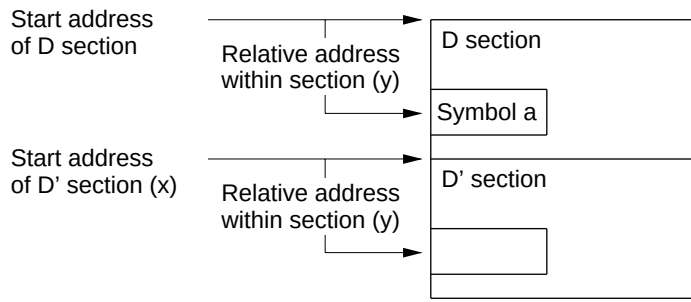


Figure 2-21 Symbol Address for Storing Program in ROM

(3) Data is copied from ROM to RAM in the start-up routine.

The copy process is included in the start-up routine. The procedure for including this process is described in the C Compiler User’s Manual.

Section 3 Executing the Linkage Editor

To execute the Linkage Editor, start Linkage Editor by entering a command line. This command line specifies the names of files to be input, and also specifies options giving various instructions to the Linkage Editor. If these instructions are sufficient, the Linkage Editor can be executed using the command line alone. If further instruction are needed, they can be given in subcommands.

Specifying Command Line: This method executes linkage by simply specifying the input files and options on the command line. It is used when only a few files are to be input and the linkage operation is relatively straightforward.

Specifying Subcommands: This method, in addition to a command line, uses subcommands to control the Linkage Editor. The subcommands specify files to be input and output, and execution control parameters for the Linkage Editor. This method is used when a large number of files or modules are specified, when the order in which sections are to be linked specified, or when multilinkage function is used. There are two ways of specifying subcommands: One is direct input from the keyboard or other input device in interactive mode and the other is input from a subcommand file.

For file name specifications, refer to appendix B, File Name Specifications. Table 3-1 shows the notes on Linkage Editor usage.

Table 3-1 Notes on Linkage Editor Usage

OS	Notes
MS-DOS	Before using this Linkage Editor, set the MS-DOS configuration file (CONFIG.SYS) with the editor as follows. `FILES=20` (1) `SHELL=a:\command.com a:\ /p` (2) 1. The number of files that is allowed to open at one time during Linkage Editor operation. 2. Directory path specification that is required when COMMAND.COM is reloaded.
UNIX	The OS shell (command interpreter) checks the command line before passing control to the Linkage Editor. Use characters that the OS allows on the command line.

3.1 Command Line Format

The following format is used for the Linkage Editor command line.

```
lnkΔ[<input file name>[{,|Δ}<input file name>]...]
  [[Δ]-<option name>[[Δ]-<option name>...]] (RET)
```

Command Name: "lnk" is input to start up the Linkage Editor.

Input File Names: Names of files to be input in the Linkage Editor are specified. These can be object module files or relocatable load module files. When more than one file is specified, the names are delimited by a comma (,).

If the file type is not specified with the input file name, the Linkage Editor automatically assumes that the type is ".obj."

Option Names: Each option name must be preceded by a hyphen (-). When an option name follows an input file name or another option name, one or more spaces or tabs can be inserted to delimit the names, or they can be entered continuously. Option names are described in detail in section 4, Linkage Editor Options and Subcommands.

Specifying the Execution Mode: Command line specification determines whether linkage is to be executed by the command line only or subcommands are to be used as well.

(a) Specifying execution by command line: If one or more input files are specified on the command line and no subcommand file is specified, module linkage will be executed according to the command line only.

(b) Specifying subcommands: If no input files are specified on the command line, or a subcommand file is specified, the Linkage Editor will be controlled by the subcommands.

3.2 Executing by Command Line

In this method, input files are specified on the command line, and the Linkage Editor executes module linkage according to the information specified in the command line alone. Output files and other instructions to the Linkage Editor are specified in the form of options. Command line execution is sufficient for performing linkage operations when the number of input files is small, and when there is no need for detailed instructions to the Linkage Editor such as the order in which sections are to be linked. Examples of execution by command line only are given below. For details on options in these examples, see section 4, Linkage Editor Options and Subcommands.

EXAMPLE 1:

```
lnk∆add,sub,mul,div∆-OUTPUT=arith∆-ENTRY=main (RET)
```

Four files “add.obj,” “sub.obj,” “mul.obj,” and “div.obj” are input to the Linkage Editor. They are linked and output as absolute load module file “arith.abs.” Export symbol “main” is the start address for execution of the output load module file. No linkage list is output.

EXAMPLE 2:

```
lnk∆main,key,display,print-OUTPUT=calc-PRINT=calc-FORM=R-DEBUG (RET)
```

Four files “main.obj,” “key.obj,” “display.obj,” and “print.obj” are input to the Linkage Editor. They are linked and output as relocatable load module file “calc.rel.” Debugging information is incorporated in this load module file. Linkage list “calc.map” is to be output.

3.3 Controlling by Subcommands

When a large number of files or modules must be input, or when complex section is linked, the command line alone may not be sufficient to contain all the specifications. In such cases, subcommands are used to control the Linkage Editor. Subcommands can be entered one at a time in interactive mode, from the keyboard or other standard input device, or a subcommand file consisting of a group of subcommands can be created in advance, and subcommands can be entered from this subcommand file.

Interactive Mode: Can be used when the number of subcommands is relatively small. This method is also useful when the Linkage Editor is employed during program debugging, where it is desired to check interim linkage results or make provisional recovery from errors.

Subcommand File: A subcommand file is used to control the Linkage Editor when the number of subcommands is large, or the procedures to be carried out are mostly routine.

A subcommand file is used by specifying the SUBCOMMAND option on the command line. The name of the subcommand file to be input is specified as a parameter of the SUBCOMMAND option.

The Linkage Editor can use a subcommand file even when subcommands are input interactively. Specify the SUBCOMMAND subcommand with the subcommand file name as a parameter.

3.3.1 Executing in Interactive Mode

In this method, subcommands required for Linkage Editor operations are input directly from the standard input device. Execution proceeds by this method when no input files are specified on the command line and the SUBCOMMAND option is not specified. Use the interface mode when the number of subcommands to be input is relatively small, or when it is desired to confirm linkage results while inputting subcommands, as in the first stage of program debugging. When the debugging support function is used, the interface mode is the most suitable.

An example showing input of subcommands in interactive mode is given below. Functions of the subcommands listed here are detailed in section 4, Linkage Editor Options and Subcommands.

EXAMPLE:

```
  lnk (RET)..............................   (1)
: INPUT∆main (RET)......................   (2)
: INPUT∆send,receive, exchange (RET).....   (3)
: INPUT∆account (RET)...................   (4)
: LIBRARY∆syslib (RET)..................   (5)
: PRINT∆ # (RET)........................   (6)
: FORM∆R (RET)..........................   (7)
: EXIT  (RET)...........................   (8)
```

(1) Command line, starting up the Linkage Editor in interactive mode.

(2) Inputs object module file “main.obj.”

(3) Inputs three object module files “send.obj,” “receive.obj,” and “exchange.obj.”

(4) Inputs object module file “account.obj.”

(5) Inputs library file “syslib.lib.”

(6) Outputs linkage list to standard output device.

(7) Creates a load module in relocatable format.

(8) Outputs load module file “main.rel” and ends the linkage operation.

3.3.2 Executing from a Subcommand File

In this method, a subcommand file is used which has been created in advance and which contains the subcommands necessary for Linkage Editor operations. This subcommand file is specified as a parameter of the SUBCOMMAND option or subcommand. This method is used when the number of subcommands to be specified is large, or the same linkage process is carried out repeatedly. It saves trouble of inputting subcommands from the keyboard one at a time.

A subcommand file is created using an editor. An example of executing from a subcommand file is given below. Functions of the subcommands listed here are detailed in section 4, Linkage Editor Options and Subcommands.

EXAMPLE 1:

```
lnkΔ-SUBCOMMAND=prglnk.sub (RET).....     (1)
```

Contents of subcommand file “prglnk.sub”:

```
OUTPUTΔfunction.......................     (2)
INPUTΔsin,cos,tan....................     (3)
INPUTΔasin,acos,atan.................     (4)
INPUTΔhsin,hcos,htan.................     (5)
INPUTΔlog,log10......................     (6)
FORMΔA...............................     (7)
EXIT.................................     (8)
```

(1) Command line, starting up the Linkage Editor and entering subcommands from subcommand file “prglnk.sub.”

(2) Names the output file as “function.” Either “.rel” or “.abs” is assumed, because the file type is omitted.

(3) Inputs object module files “sin.obj,” “cos.obj,” and “tan.obj.”

(4) Inputs object module files “asin.obj,” “acos.obj,” and “atan.obj.”

(5) Inputs object module files “hsin.obj,” “hcos.obj,” and “htan.obj.”

(6) Inputs object module files “log.obj” and “log10.obj.”

(7) Creates a load module in absolute format. The file type for the output file name becomes “.abs.”

(8) Outputs load module file “function.abs” and ends the linkage operation.

EXAMPLE 2:

```
  lnk (RET)............................ (1)
: SUBCOMMAND pgmlnk.sub (RET)......... (2)
```

(1) Command line, starting up the Linkage Editor. Module linkage is executed interactively, because no parameters are specified.

(2) Inputs subcommands from “pgmlnk.sub.”

If there is no EXIT subcommand in the subcommand file, the Linkage Editor waits for further subcommand input.

3.4 Terminating the Linkage Editor

When terminated, the Linkage Editor returns an error level to the system as a return code. This return code controls the execution of a command file.

The return code has the values shown in table 3-2, depending on the error level.

Table 3-2 Return Code Depending on Error Level

	Return Code	
Error Level	**MS-DOS**	**UNIX**
Normal termination	0	0
Warning	0	0
Error	2	1
Fatal error	4	1

Section 4 Linkage Editor Options and Subcommands

Options and subcommands specify file names and give the Linkage Editor various instructions, such as the order in which sections are to be linked. Options and subcommands have four types of functions: file control, memory allocation, execution control, and debugging support. These functions can be used independently or in combination to edit load modules in various ways.

(1) File Control Functions: File control functions specifies input files and output files to the Linkage Editor. Input files include object module files, relocatable load module files and library files. Output files are load module files and list files.

(2) Memory Allocation Functions: Memory allocation functions can inform the Linkage Editor the order in which sections are to be linked and give their start addresses. They can also specify the address at which the output load module is to start executing. These functions change the order in which sections are linked, or create a load module that is to execute from a specified address.

(3) Execution Control Functions: Execution control functions specify the form in which the Linkage Editor is to input and output information, and end Linkage Editor operations. They input subcommands from a subcommand file, or incorporate debugging information in a load module.

(4) Debugging Support Functions: Debugging support functions display contents of a load module during a linkage operation, or change information such as export and import symbol names, etc. These are useful at the program debugging stage, for confirming interim linkage results, or for provisional recovery from errors.

Options and subcommands have the same names and have equivalent functions, but are specified using different formats. Moreover, some specifications can be made only with either subcommands or options. Section 4.1, Option and Subcommand Formats, and section 4.2, List of Options and Subcommands, should accordingly be read carefully.

For details on the functions and means of specifying each option and subcommand, refer to sections 4.3, File Control, through 4.6, Debugging Support.

4.1 Option and Subcommand Formats

(1) Option and Subcommand Structure:

(a) Name: The name part gives the name of the option or subcommand. For details, see section 4.2, List of Options and Subcommands.

(b) Parameters: The parameter part gives information such as the name of files on which the option or subcommand operates, and address values. There are different requirements and methods of specification depending on the option or subcommand. See sections 4.3, File Control, 4.4, Memory Allocation, 4.5, Execution Control, and 4.6, Debugging Support.

Options and subcommands differ as to the way of separating the name from the parameters. Options use an equals sign (=), while subcommands use one or more spaces or tabs.

Option format

```
<Name>=<parameters>
```

Subcommand format

```
<Name>Δ<parameters>
```

EXAMPLES:

```
-OUTPUT=loadf..........
OUTPUTΔloadf...........
```

Option

Subcommand

In these examples, “OUTPUT” is the name, and “loadf” is the parameter.

(2) Continuation Specification for a Subcommand: When a subcommand is too long to be specified on one line (generally, up to 500 characters per line, but it will depend on the OS), a continuation specifier is used. This is an ampersand (&) at the end of the line. It must always be placed in between two parameters; if it is placed within a parameter, it will be interpreted as part of the parameter. If a character (other than a space or tab) is typed after the ampersand, an error will occur and the subcommand will not be continued.

If continuation is specified in interactive mode, a hyphen (-) appears as a prompt for further input.

EXAMPLES:

```
:INPUTΔ obj00,lib(mod0,mod1),& (RET)
-obj01, obj02 (RET)
```

Continuation specifier

```
:INPUTΔ obj00,lib(mod0,mod1),ob& (RET)
:
```

Not a continuation line

Processed under the file name ob& due to specification within parameter

(3) Specifying Comments in a Subcommand File: A comment specifier adds notes or other comments in a subcommand file. The specifier is a semicolon (;) placed on a subcommand line, indicating that the rest of the line is a comment. At least one space or tab must set off the semicolon from the subcommand name or parameter.

When a semicolon is placed at the beginning of a subcommand line, the entire line is taken as a comment.

EXAMPLES:

```
; EXAMPLE OF LINKAGE SUBCOMMAND
```

..... The entire line is a comment.

```
LIBRARYΔsyslibΔ; INDICATES LIBRARY FILE
```

..... “INDICATES LIBRARY FILE” is a comment.

```
INPUTΔobject.rel;abc
```

..... object.rel;abc” is treated as one parameter.

4.2 List of Options and Subcommands

There are 20 options and 29 subcommands. The options and subcommands are listed in table 4-1.

Options and subcommands can be written either in uppercase or lowercase letters.

Table 4-1 List of Options and Subcommands

No.	Type	Option/ Subcommand Name	Function	Option	Sub-command	Section
1	File control	INPUT	Specifies input file	No	Yes	4.3.1
		OUTPUT* (NOOUTPUT)	Specifies output file	Yes	Yes	4.3.2
		LIBRARY (NOLIBRARY)*	Specifies library file	Yes	Yes	4.3.3
		PRINT (NOPRINT)*	Specifies list file	Yes	Yes	4.3.4
		EXCLUDE (NOEXCLUDE)*	Excludes modules from linking	Yes	Yes	4.3.5
		DIRECTORY	Specifies directory name replacement	No	Yes	4.3.6
2	Memory allocation	START	Specifies section start address and linking order	Yes	Yes	4.4.1
		ENTRY	Specifies execution start address	Yes	Yes	4.4.2
		ALIGN_SECTION	Specifies linkage of sections having different boundary alignment values	Yes	Yes	4.4.3
		CHECK_SECTION	Specifies section check	Yes	Yes	4.4.4
		AUTOPAGE (NOAUTOPAGE)*	Specifies automatic paging	Yes	Yes	4.4.5
		CPU	Specifies address check	Yes	Yes	4.4.6
		CPUCHECK	Specifies output of errors at address check	Yes	Yes	4.4.7
		ROM	Specifies support of storing program in ROM	Yes	Yes	4.4.8

Notes:
1. The shortest permissible abbreviated forms are underlined.
2. Yes and No in the table indicate whether an item can be used as an option or subcommand.
3. An asterisk indicates the default option or subcommand.

Table 4-1 List of Options and Subcommands (cont)

No.	Type	Option/ Subcommand Name	Function	Option	Sub-command	Section
3		EXCHANGE	Substitutes units	No	Yes	4.5.1
		SUBCOMMAND	Specifies subcommand file	Yes	Yes	4.5.2
		FORM	Specifies format of output load module file	Yes	Yes	4.5.3
		DEBUG (NODEBUG)*	Specifies output of debugging information	Yes	Yes	4.5.4
		SDEBUG	Specifies output of debugging information to a file	Yes	Yes	4.5.5
		END	Terminates subcommand input	No	Yes	4.5.6
		EXIT	Terminates linkage operation	No	Yes	4.5.7
		ABORT	Aborts linkage operation	No	Yes	4.5.8
		ECHO* (NOECHO)	Specifies subcommand file echo-back	Yes	Yes	4.5.9
		UDF* (NOUDF)	Specifies display of undefined symbols	Yes	Yes	4.5.10
		UDFCHECK	Specifies output of error for undefined symbol	Yes	Yes	4.5.11
4	Debugging support	LIST	Displays interim linkage information	No	Yes	4.6.1
		RENAME	Changes name of unit, export symbol, or import symbol	No	Yes	4.6.2
		DELETE	Deletes unit or export symbol	No	Yes	4.6.3
		DEFINE	Forcibly defines import symbol	Yes	Yes	4.6.4

Notes:
1. The shortest permissible abbreviated forms are underlined.
2. Yes and No in the table indicate whether an item can be used as an option or subcommand.
3. An asterisk(*) indicates the default option or subcommand.

(1) Negative Form of Options and Subcommands: For some options and subcommands, a negative form starting with “NO” can be specified. Parameters cannot be specified with negative-form options and subcommand. There are eight negative option/subcommand forms, as follows:

(a) NOOUTPUT: Suppresses output of load module file
(b) NOLIBRARY: Specifies non-use of a library file
(c) NOPRINT: Suppresses output of a list file
(d) NOEXCLUDE: Specifies linking of modules
(e) NOAUTOPAGE: Suppresses automatic paging
(f) NODEBUG: Suppresses output of debugging information
(g) NOECHO: Suppresses echo-back of a subcommand file
(h) NOUDF: Suppresses display of undefined symbols

(2) Option Default: When an option is omitted, the following are the default choices.

(a) OUTPUT (no parameters)
(b) NOLIBRARY
(c) NOPRINT
(d) NOEXCLUDE
(e) NOAUTOPAGE
(f) FORM=A
(g) NODEBUG
(h) ECHO
(i) UDF

(3) Abbreviating Option and Subcommand Names: Names of options and subcommands can be abbreviated to the point where the name can still be distinguished from other names. For example, consider the name “DEBUG.”

D: Cannot be distinguished from DELETE or DEFINE, so an error occurs
DE: Cannot be distinguished from DELETE or DEFINE, so an error occurs
DEB: Recognized as DEBUG
DEBU: Recognized as DEBUG
DEBUG: Recognized as DEBUG
DEBUGS: No such name, so an error occurs

(4) Range of Validity of Options: When only a command line is specified, linkage is executed based only on the options specified. When subcommands are specified, options specified in the command line remain valid up to the first END subcommand specified (or up to the EXIT subcommand when no END is specified). However, if subcommands are specified which conflict with the function of an option, an error message is displayed, the option becomes invalid, and execution proceeds according to the subcommand specification. After the first END subcommand, all subsequent subcommand specifications are valid.

EXAMPLE:

```
 lnk∆ -NOOUTPUT (RET)
      ⋮
:END (RET)
      ⋮
:OUTPUT∆ loadfile (RET)
      ⋮
```

The NOOUTPUT option is in effect, so no output file is created.

The OUTPUT subcommand is now valid, so output file “loadfile.abs” is created.

In the following sections the format below is used to describe each option and subcommand.

Heading for each option or subcommand

No. **INPUT**

Format Name Option Subcommand Negative Form

Parameters

Option or subcommand name, and format for specifying parameters; underline indicates shortest abbreviation

Function

Summary of option or subcommand functions

Explanation

Detailed description of functions, and restrictions

Examples

Examples of option or subcommand specifications

4.3 File Control

4.3.1 INPUT—Specifies Input Files INPUT

Format	Name	Option	Subcommand	Negative Form
		None	INPUT	None
	Parameters	<Input file name>[(<module name>[,<module name>...])] [{,|Δ}<Input file name> [(< module name>[,<module name>...])]...]		

Function Specifies files and modules to be input.

Explanation **(1) Outline of functions:**

- The files specified by parameters, or the specified modules in those files, are input to the Linkage Editor.
- Three kinds of files can be specified: object module files, load module files, and library files.
- Modules can be specified only for library files, in which case only the specified modules from the library file will be input.
- If the file type is omitted from a file name, the Linkage Editor will automatically assume the type as follows.

  No module name specified: ".obj"

  Module name specified: ".lib"

(2) Restrictions in use:

- Among load module files, only relocatable load modules can be specified. If an absolute load module is specified, an error will occur and the file will not be input.
- If a module other than that in a library file is specified, an error will occur and the file will not be input.
- The maximum number of input files that can be treated in one linkage process is 256, including library files. If more than 256 files are specified, an error will occur, and only the first 256 files specified will be input. To process more than 256 files, use the multilinkage function.
- Page type and non-page type modules must not be input at the same time. If both types of modules are input together, an error will occur and the Linkage Editor will stop execution.

Examples

`INPUT∆main`

Inputs the object module file “main.obj.”

`INPUT∆funclib(sin,cos),tan.o`

Inputs the modules “sin” and “cos” from library file “funclib.lib,” and inputs the object module file “tan.o.”

Format

Name	Option	Subcommand	Negative Form
	OUTPUT	OUTPUT	NOOUTPUT
Parameters	[<Output file name>]		

Function

Specifies a load module output file name.

Explanation

(1) Outline of functions:

- Outputs the load module generated by the Linkage Editor to the specified file.
- If the file type is omitted from the file name, the Linkage Editor will automatically assign a file type according to the format of the load module file, as follows.

  Absolute format “.abs”

  Relocatable format “.rel”

  The format of the load module file is specified using the FORM option or subcommand. If no specification is made, absolute format is used.
- If no output file name is specified using the OUTPUT option or subcommand, the output file is given the name of the first specified input file plus the above file type.
- If the NOOUTPUT option or subcommand is specified, no load module file will be output.

(2) Restrictions in use:

- No parameters can be specified with the NOOUTPUT option or subcommand.
- If an output file name is specified, it must be different from all input file names.

Examples

`-OUTPUT=prgload`

Outputs load module file “prgload.abs” (or “prgload.rel”).

`-OUTPUT`

Outputs load module file with the name of the first specified object module file plus “.abs” (or “.rel”).

`OUTPUT∆main.10`

Outputs load module file “main.10.”

Format

Name	Option	Subcommand	Negative Form
	LIBRARY	LIBRARY	NOLIBRARY
Parameters	<Library file name>[,<library file name>...]		

Function

Specifies input library files.

Explanation

(1) Outline of functions:

- Specifies library files which the Linkage Editor is to search if there are unresolved import symbols after linkage operations among specified input files are completed.
- If both user library files and system library files are specified, the Linkage Editor will search the user library files first.
- If no file type is specified with the library file name, the Linkage Editor automatically assumes this to be ".lib."
- If the NOLIBRARY option or subcommand is specified, there will be no input from a library file (including default libraries). When linkage is controlled by subcommand specification, however, the range of validity of this option is limited. For details see Range of Validity of Options under section 4.2.

(2) Restrictions in use:

- Only library files created using the H Series Librarian can be input to the Linkage Editor.
- The maximum number of input files that can be treated in one linkage operation is 256, including library files. If more than 256 files are specified, an error will occur, and only the first 256 files specified will be input. To process more than 256 files, use the multilinkage function.
- Page type and non-page type modules must not be input at the same time. If both types of modules are input together, an error will occur and the Linkage Editor will stop execution.
- No parameters must be specified with the NOLIBRARY option or subcommand.

Examples

`-LIBRARY=syslib.`

Specifies library file "syslib."

`LIBRARY∆system,debug`

Specifies library files "system.lib" and "debug.lib."

Format

Name	Option	Subcommand	Negative Form
	PRINT	PRINT	NOPRINT
Parameters	<List file name> #		

Function

Specifies a list file for output of linkage list.

Explanation

(1) Outline of functions:

- Outputs a linkage list to the specified list file.
- If the parameter "#" is specified, the list file is output to the standard output device.
- If no PRINT option or subcommand is specified, or if the NOPRINT option or subcommand is specified, the linkage list will not be output.
- If no file type is specified with the list file name, the Linkage Editor will automatically assume this to be ".map."
- On the contents of the linkage list, see section 6.1, Linkage Lists.

(2) Restrictions in use:

- No parameters must be specified with the NOPRINT option or subcommand.

Examples

```
-PRINT=linkage
```

Outputs a linkage list to list file "linkage.map."

```
PRINT∆earth.prn
```

Outputs a linkage list to list file "earth.prn."

Format

Name	Option	Subcommand	Negative Form
	EXCLUDE	EXCLUDE	NOEXCLUDE
Parameters	None		

Function

Specifies that modules defining non-referenced import symbols should not be linked.

Explanation

(1) Outline of functions:

- If an import symbol is not referenced, the module defining it is not linked.
- When the NOEXCLUDE option or subcommand is specified, modules defining non-referenced import symbols are linked. The defining modules are also linked if the EXCLUDE option or subcommand is omitted.

(2) Restrictions in use:

- The EXCLUDE subcommand cannot be used after input files have been specified by the INPUT or EXCHANGE subcommand.
- The EXCLUDE option or subcommand can be specified only when the output load module is in absolute format. When the multilinkage function is used to create an absolute load module in the final linkage process, if the default library function is also used, the modules from the default library will be included in the first linkage process. If you want the default library to be included in the last linkage process, specify the NOLIBRARY subcommand for the intermediate linkage processes.

Examples

```
-EXCLUDE
```

If an import symbol is not referenced, the module defining it is not linked.

Format

Name	Option	Subcommand	Negative Form
	None	DIRECTORY	None
Parameters	<Symbol name>(<Directory name>)		

Function

Defines a symbol as an alias of a directory. This function enables a long directory name to be input with a simple symbol name.

Explanation

- Directory name alias definition

  A symbol name is defined as an alias of a directory with the DIRECTORY subcommand.

  DIRECTORYΔ<symbol name>(<directory name>)

- Directory name reference

  To refer to a directory name, enclose the defined symbol name with a dollar mark ($) and a slash (/) (a dollar mark ($) and a back-slash (\) in MS-DOS system). If the symbol name has not been defined, the Linkage Editor does not replace it with a directory name.

  $<symbol name>/ —> Replaced with <directory name>/

- Symbol names for up to 16 directory names can be defined.

Examples

```
DIRECTORYΔsymbol(dir1/dir2)
INPUTΔ$symbol/file1.obj
```

Defines symbol “symbol” as an alias of directory “dir1/dir2”.

Replaces $symbol/with dir1/dir2, and as a result, specifies file name “dir1/dir2/file1.obj”.

4.4 Memory Allocation

4.4.1 START—Specifies Start Address and Linkage Order of Sections START

Format

Name	Option	Subcommand	Negative Form
	START	START	None

Parameters

Option

UNIX: <Section name>[,<section name>…][/[<page address>:]<start address>]
[,<section name>[,<section name>…][/[<page address>:]<start address>]...]

MS-DOS: <Section name>[,<section name>…][([<page address>:]<start address>)]
[,<section name>[,<section name>…][([<page address>:]<start address>)]...]

Subcommand: <Section name>[,<section name>…][([<page address>:]<start address>)]
[,<section name>[,<section name>…][([<page address>:]<start address>)]...]

Function

Specifies the order in which sections are linked, and their start addresses.

Explanation

(1) Outline of functions:

- Sections are allocated from the specified address and in the specified order.
- If the start address is not specified and only the section linkage order is specified, and sections are assigned addresses starting from zero.
- Page address can be specified only for page type modules. If the page address is not specified, it is assumed to be zero.
- The page address and start address are specified in hexadecimal notation.
- When sections not specified in the parameters are input, those sections are assigned after the series of sections with the highest specified start address.
- If no START option or subcommand is specified, sections will be allocated to addresses starting from zero in the order of appearance.
- The START option or subcommand can be specified more than once.
- Hexadecimal numbers must start with numbers 0 through 9.

(2) Restrictions in use:

- If the load module to be output is in relocatable format, the START option or subcommand must not be used.
- If a page address is specified for non-page type modules, an error will occur and the Linkage Editor will stop execution.

  EX: 0ABCD Correct designation

  ABCD Incorrect designation

Explanation

- Page addresses must be assigned in the range from 0 through 0FF (hexadecimal).
- The range of start addresses that can be specified varies with the H series model.

H8/500 series:	0 through 0FFFF (hexadecimal)
H8/300 series:	300HA: 0 through 0FFFFFF (hexadecimal) Others: 0 through 0FFFF (hexadecimal)
H8/S series:	2600A and 2000A: 0 through 0FFFFFFFF (hexadecimal) Others: 0 through 0FFFF (hexadecimal)
SH series:	0 through 0FFFFFFFF (hexadecimal)

Examples

`-START=CODE,DATA,BSS,STACK`

Links sections in the order “CODE,” “DATA,” “BSS,” “STACK,” and allocates them to addresses starting from 0 (hexadecimal)

`-START=CONTROL,BANK0,BANK1(0F00) (MS-DOS)`

`-START=CONTROL,BANK0,BANK1/0F00 (UNIX)`

Links sections in the order “CONTROL,” “BANK0,” “BANK1,” and allocates them to addresses starting from 0F00 (hexadecimal).

`START∆CONTROL,BANK0,BANK1(0:0F00)`

Links sections in the order “CONTROL,” “BANK0,” “BANK1,” and allocates them to addresses starting from 0F00 (hexadecimal) in page 0.

`START∆RAM0,RAM1(8000),ROM1,ROM2(1000),ROM0`

Links sections “RAM0” and “RAM1” in that order and allocates them to addresses starting from 8000 (hexadecimal). Sections “ROM1” and “ROM2” are linked in that order and are allocated to addresses starting from 1000 (hexadecimal). Section “ROM0” is allocated to addresses starting from zero.

Format

Name	Option	Subcommand	Negative Form
	ENTRY	ENTRY	None
Parameters	<Export symbol>		

Function

Specifies the start address for executing a load module.

Explanation

(1) Outline of functions:

- Sets the address of an export symbol as the execution start address of a load module to be output.
- If no ENTRY option or subcommand is specified and the output load module format is absolute, the execution start address becomes the start address of the first code section in the output load module.

(2) Restrictions in use:

- If an ENTRY option or subcommand is specified more than once, the last specified address is valid.

Examples

```
-ENTRY=PRG_ENT
```

Specifies the address of export symbol “PRG_ENT” as the execution start address.

```
ENTRY∆MAIN
```

Specifies the address of export symbol “MAIN” as the execution start address.

4.4.3 ALIGN_SECTION—Specifies Linkage of Sections Having Different Boundary Alignment Values

Format

Name	Option	Subcommand	Negative Form
	ALIGN_SECTION	ALIGN_SECTION	None
Parameters	None		

Function

Specifies address assignment for sections having the same name but different boundary alignment values (specified with the ALIGN operand in the .SECTION directive of the assembler), handling the sections as the same one.

Explanation

Outline of functions:

- Sections having the same name but different boundary alignment values can be generated by using the ALIGN operand in the .SECTION directive of the assembler. In this case, the Linkage Editor usually does not handle these sections as the same section when assigning addresses because they have different boundary alignment values. Specifying the ALIGN_SECTION option enables these sections to be handled as the same section.

Examples

```
-ALIGN_SECTION
```

Assigns addresses for sections having different boundary alignment values handling the sections as the same section.

Format

Name	Option	Subcommand	Negative Form
	CHECK_SECTION	CHECK_SECTION	None
Parameters	None		

Function

Outputs a warning and continues processing if a section that has not been specified with the START option/subcommand is found in an input file.

Explanation

(1) Outline of functions:

- Checks whether the input files include a section whose start address has not been specified with the START option/subcommand, and outputs warning message 120 when such a section is found.

(2) Restrictions in use:

- Processing continues after the warning message is output.

Examples

```
-CHECK_SECTION
```

Checks whether the input files include a section whose start address has not been specified and outputs a warning when such a section is found.

Format

Name	Option	Subcommand	Negative Form
	AUTOPAGE	AUTOPAGE	NOAUTOPAGE
Parameters	None		

Function

Specifies autopaging in assignment of addresses to page type modules.

Explanation

(1) Outline of functions:

- When a page type module is linked, addresses are assigned by automatic paging.
- If the AUTOPAGE option or subcommand is not specified, or if the NOAUTOPAGE option or subcommand is specified, addresses are not assigned by automatic paging.

(2) Restrictions in use:

- The AUTOPAGE option or subcommand must not be specified when linking non-page type modules are linked. Such specification will result in an error, and the Linkage Editor will stop execution.
- If the NOAUTOPAGE option or subcommand is specified when page type modules are linked, sections may overlap page boundaries. If overlap occurs, the Linkage Editor displays a warning.

Examples

```
AUTOPAGE
```

Assigns addresses by autopaging.

```
-NOAUTOPAGE
```

Assigns addresses without regard to page boundaries.

Format

Name	Option	Subcommand	Negative Form
	CPU	CPU	None
Parameters	<CPU information file name>		

Function Specifies execution of an address check using a CPU information file.

Explanation

(1) Outline of functions:

- The validity of addresses assigned to each section is checked, based on CPU information. In the following cases the section address assignment is regarded as invalid, and the Linkage Editor displays a warning. The sections, however, are output to the load module file without changing the addresses.
  - (a) When sections are assigned addresses in areas other than memory.
  - (b) When one section is assigned to addresses overlapping memory areas having different memory types and attributes.
- If no file type is specified with the CPU information file, the Linkage Editor will automatically assume this to be ".cpu."

(2) Restrictions in use:

- In the following cases the Linkage Editor displays a warning, and the CPU option or subcommand is invalid.
  - (a) Relocatable format is specified for load module output with the FORM option or subcommand.
  - (b) The information format of the CPU information file is invalid.
  - (c) A CPU information file is specified for linkage processing of object modules that are not for the H8S, H8/300, or SH series.
- When a CPU option or subcommand is specified more than once, a warning message is displayed, and only the last-specified file is valid.

Examples

`-CPU=cinf`

Inputs CPU information file "cinf.cpu."

`CPU∆c300.inf`

Inputs CPU information file "c300.inf."

4.4.7 CPUCHECK—Specifies Error Output at Address Check Using CPU Information File

Format

Name	Option	Subcommand	Negative Form
	CPUCHECK	CPUCHECK	None
Parameters	None		

Function

Changes the warning message into an error message when an address check is executed with the CPU option/subcommand using the CPU information file.

Explanation

(1) Outline of functions:

- Outputs error 329 and aborts processing when memory allocation does not match the memory layout specified in the CPU information file. This error occurs in the same conditions as those generating a warning when the CPU option/subcommand is specified (see section 4.4.6).

(2) Restrictions in use:

- When neither the CPU option nor subcommand is specified, the CPUCHECK option/subcommand is ignored.

Examples

```
-CPUCHECK
```

Specifies error message output in the conditions that generate a warning at CPU option/subcommand execution and aborts processing in these cases.

Format

Name	Option	Subcommand	Negative Form
	ROM	ROM	None

Parameters

UNIX: <Section 1>/<Section 2>[,<Section 1>/<Section 2>…]
MS-DOS: (<Section 1>,<Section 2>)[(<Section 1>,<Section 2>),…]

<Section 1>: Section name of source initialized data area in ROM
<Section 2>: Section name of destination initialized data area in RAM

Function

Reserves a RAM area for updating initialized data values stored in ROM.

Explanation

(1) Outline of functions:

- In the output load module, a section with the same section size as the specified section 1 is reserved as section 2. Section 2 has the same section attributes as section 1.
- References to symbols declared in section 1 are relocated to addresses in section 2. Specify a relocatable section as section 1.
- Up to 64 pairs of section 1 and section 2 pairs can be specified.
- For details of the support of storing program in ROM, see section 2.7, Support of Storing Program in ROM.

(2) Restrictions in use:

- The ROM option or subcommand cannot be specified when the output load module has the relocatable format.
- If two sections have the same name and this name is specified as section 1, the section input first is selected.
- An error occurs if section 1 does not exist.
- A dummy section cannot be specified as section 1.
- When an existing section is specified as section 2, the following conditions must be satisfied.
  (a) The size of section 2 in each unit is 0.
  (b) Section 2 is the relocatable section.
  (c) Both section 1 and section 2 have the same attribute.

Examples

```
-ROM=D/RAM_SCT (UNIX)
-ROM=(D,RAM_SCT) (MS-DOS)
```

Reserves section RAM_SCT, equal in size to section D, in the output load module. References to symbols allocated to section D are relocated to addresses on RAM_SCT.

4.5 Execution Control

4.5.1 EXCHANGE—Forcibly Replaces Units EXCHANGE

Format

Name	Option	Subcommand	Negative Form
	None	EXCHANGE	None
Parameters	<Input file name>[(<unit name>[,<unit name>...])]		

Function

Replaces units in an input file by units of the same name in the load module being processed by the Linkage Editor.

Explanation

(1) Outline of functions:

- Units in the specified input file are replaced by units of the same name in the load module being processed by the Linkage Editor.
- An object module file or load module file can be specified as the input file.
- If a load module is specified as the input file without specifying unit names, all the units in that load module file will be usable for replacement.
- If no file type is given with the input file name, the Linkage Editor will automatically assume ".obj" as the file type.
- Units are replaced after all input files have been included. If more than one EXCHANGE subcommand is specified, units will be replaced in the order of specification.

(2) Restrictions in use:

- An absolute load module must not be specified. If an absolute load module is specified, an error will occur, and the file will not be input.
- A library file must not be specified as the input file. If a library file is specified, an error will occur, and the file will not be input.

Examples

`EXCHANGE∆datain`

Replaces units in the object module file "datain.obj" by units of the same name in the load module file being processed.

`EXCHANGE∆function.rel(tan,atan)`

Replaces the units "tan" and "atan" in relocatable load module file "function.rel" by units of the same name in the load module file being processed.

Format

Name	Option	Subcommand	Negative Form
	SUBCOMMAND	SUBCOMMAND	None
Parameters	<Subcommand file name>		

Function

Specifies a subcommand file for input.

Explanation

(1) Outline of functions:

- Subcommands are input from the specified subcommand file.
- If the SUBCOMMAND option is not specified on the command line, and no input file is specified there, the Linkage Editor will link modules according to the subcommands input in interactive mode.
- If the SUBCOMMAND option is not specified on the command line but one or more input files are specified there, the Linkage Editor will link modules according to the command line specification.

(2) Restrictions in use:

- When a subcommand file is specified on the command line together with input files or other options, the subcommand file is executed as the last option, regardless of its specification position. For example:

```
lnk in1,in2 - SUB = linkage.sub - FORM = R
      (1)          (2)              (3)
```

  This command line is interpreted and executed in the order (3), (1), (2). If FORM=A is specified in linkage.sub, FORM=A is valid (because it is interpreted afterward).
- The SUBCOMMAND subcommand cannot be specified in a subcommand file.

Examples

```
-SUBCOMMAND=linkage.sub
```

Inputs subcommand file “linkage.sub” and links modules according to the contents of this file.

Format

Name	Option	Subcommand	Negative Form
	FORM	FORM	None
Parameters	A R		

Function

Specifies the output load module file format as either absolute or relocatable.

Explanation

(1) Outline of functions:

- If parameter “A” is specified, the load module file will be output in absolute format.
- If parameter “R” is specified, the load module file will be output in relocatable format.
- If no FORM option or subcommand is specified, the load module will be output in absolute format.

(2) Restrictions in use:

- The parameter “R” cannot be specified when the ROM or START option or subcommand is specified.

Examples

```
-FORM=R
```

Outputs the load module file in relocatable format.

```
FORM∆A
```

Outputs the load module file in absolute format.

Format

Name	Option	Subcommand	Negative Form
	DEBUG	DEBUG	NODEBUG
Parameters	None		

Function Specifies incorporation of debugging information in the output load module file.

Explanation

(1) Outline of functions:

- Incorporates debugging information in the output load module file. This information is required for symbolic debugging using the Simulator/Debugger.
- If no DEBUG option or subcommand is specified, or if the NODEBUG option or subcommand is specified, debugging information will not be incorporated in the output load module file.

(2) Restrictions in use:

- If the NOOUTPUT option or subcommand is specified, the DEBUG option or subcommand is ignored.

Examples

```
DEBUG
```

Incorporates debugging information in the output load module file.

```
-NODEBUG
```

Does not incorporate debugging information in the output load module file.

Format

Name	Option	Subcommand	Negative Form
	SDEBUG	SDEBUG	None
Parameters	None		

Function

Outputs a debugging information file separately from a load module. Some debuggers require the object and debugging information as separate files. In this case, the SDEBUG option/subcommand must be specified.

Explanation

(1) Outline of functions:

- Outputs a debugging information file separately from a load module.

  Object file: File extension .abs.

  Debugging file: File extension .dbg.
- When the debugging information is output as a separate file, the time for downloading the load module at debugging can be reduced.

(2) Restrictions in use:

- When the relocatable format is specified for the output load module, the SDEBUG option/subcommand cannot be used.
- If the NOOUTPUT option/subcommand is specified, the SDEBUG option/subcommand is ignored.

Examples

```
-SDEBUG
```

Outputs a debugging file and an object file separately.

Format

Name	Option	Subcommand	Negative Form
	None	END	None
Parameters	None		

Function

Temporarily ends input of subcommands and begins linkage operation (after which subcommand input is resumed).

Explanation

(1) Outline of functions:

- Temporarily ends input of subcommands and begins a linkage operation. After the linkage operation is completed, the Linkage Editor is initialized and subcommand input is resumed.
- When the multilinkage function is used to perform multiple linkage operations during a course of Linkage Editor execution, the END subcommand indicates the end of one linkage process.
- When the multilinkage function is not used, or when the end of the final linkage process is specified in a multilinkage operation, use the EXIT subcommand in place of the END subcommand.

(2) Restrictions in use:

- If, for a single linkage process, the END subcommand is specified without specifying input files, an error will occur.

Examples

```
END
```

Temporarily ends subcommand input and begins a linkage operation.

Format

Name	Option	Subcommand	Negative Form
	None	EXIT	None
Parameters	None		

Function

Ends subcommand input and begins linkage operation (subcommand input is not resumed).

Explanation

Outline of functions:

- Ends subcommand input and begins linkage operation. After the linkage operation is completed, ends the Linkage Editor execution.
- When execution is controlled from a subcommand file, if no EXIT subcommand is specified, the Linkage Editor waits for further subcommand input.
- If, for a single linkage process, the EXIT subcommand is specified without specifying input files, an error will occur.

Examples

```
EXIT
```

Ends subcommand input and begins linkage operation.

Format

Name	Option	Subcommand	Negative Form
	None	ABORT	None
Parameters	None		

Function

Forcibly ends linkage operation.

Explanation

Outline of functions:

- Forcibly ends Linkage Editor operation.
- The ABORT subcommand is useful to interrupt Linkage Editor operation when a mistake such as subcommand input mistake has been made.

Examples

```
ABORT
```

Brings Linkage Editor execution to a forced end.

Format

Name	Option	Subcommand	Negative Form
	ECHO	ECHO	NOECHO
Parameters	None		

Function

Specifies whether or not to suppress echo-back of subcommands when a subcommand file is executed.

Explanation

Outline of functions:

- The ECHO option or subcommand displays subcommands on the console when a subcommand file is executed. Subcommands are displayed even if the ECHO option or subcommand is not specified.
- The NOECHO option or subcommand suppresses display of subcommands on the console when a subcommand file is executed.

Examples

```
-ECHO
```

Displays executed subcommands on the console when a subcommand file is executed.

Format

Name	Option	Subcommand	Negative Form
	UDF	UDF	NOUDF
Parameters	None		

Function

Specifies whether to display a warning message when an undefined symbol remains.

Explanation

(1) Outline of functions:

- Warning message 105 is displayed if an undefined symbol remains when a relocatable load module is created. This message is also displayed if an undefined symbol remains when the UDF option or subcommand is omitted.
- When the NOUDF option or subcommand is specified, a warning message is not displayed if there is an undefined symbol when a relocatable load module is created.

(2) Restrictions in use:

- The NOUDF option or subcommand is ignored when an absolute load module is created.

Examples

```
-FORM=R-NOUDF
```

Does not display a warning message if there is an undefined symbol when the relocatable load module is created.

Format

Name	Option	Subcommand	Negative Form
	UDFCHECK	UDFCHECK	None
Parameters	None		

Function

Displays an error message for an undefined symbol and stops absolute load module generation.

Explanation

(1) Outline of functions:

- Outputs error message 221 and stops absolute load module generation when an undefined import symbol is found. (When the UDFCHECK is not specified, warning message 105 is output instead and absolute load module generation continues.)

(2) Restrictions in use:

- When relocatable load module generation is specified, the UDFCHECK option/subcommand is ignored.

Examples

```
-UDFCHECK
```

Displays an error message for an undefined symbol and stops absolute load module generation.

4.6 Debugging Support

4.6.1 LIST—Displays Interim Linkage Information LIST

Format

Name	Option	Subcommand	Negative Form
	None	LIST	None
Parameters	{ M / U / X }		

Function Displays linkage information of an input file.

Explanation

(1) Outline of functions:

- Outputs linkage information to the standard output device concerning the files currently being input.
- Content of the displayed information depends on the specified parameters, as follows.

  M: Displays a link map

  U: Displays unresolved import symbols

  X: Displays export symbols

(2) Restrictions in use:

- To display linkage information according to the input files, the information displayed is restricted as follows.

  — When parameter M is specified

  The start address of a relocatable section is always 0.

  — When parameter U is specified

  The display shows import symbols for which there is no corresponding export symbol in the input files specified in INPUT subcommands up to the location of the LIST subcommand.

Examples

`LISTΔM`

Displays a linkage map for the load module being processed.

`LISTΔU`

Displays unresolved import symbols in the load module being processed.

4.6.2 RENAME—Changes the Names of Units, Export Symbols, or Import Symbols

Format

Name	Option	Subcommand	Negative Form
	None	RENAME	None

Parameters

```
{ UN=<unit name 1> (<unit name 2>)
  ER=<unit name>.<import symbol 1>
      (<import symbol 2>)
  ED=<unit name>.<export symbol 1>
      (<export symbol 2>)
}
[ , { UN=<unit name 1>(<unit name 2>)
      ER=<unit name>.<import symbol 1>
          (<import symbol 2>)
      ED=<unit name>.<export symbol 1>
          (<export symbol 2>)
    } ... ]
```

Function

Changes the names of units, export symbols or import symbols in input files.

Explanation

(1) Outline of functions:

- Changes the names of the specified units, export symbols, or import symbols in input files to the name designated in parentheses (“( )”).
- The unit name specified following “UN=” is changed to the unit name in parentheses.
- The import symbol name specified following “ER=” is changed to the name in parentheses. The import symbol name is preceded by the name of the unit in which the symbol exists, and is set off from the unit name by a period (.).
- The export symbol name specified following “ED=” is changed to the name in parentheses. The export symbol name is preceded by the name of the unit in which the symbol exists, and is set off from the unit name by a period (.).

Explanation **(2) Restrictions in use:**

- The RENAME subcommand will affect the input files specified only in the first INPUT subcommand after the RENAME subcommand.
- Only the following five subcommands can be specified immediately after the RENAME subcommand:

  (a) INPUT subcommand
  (b) EXCHANGE subcommand
  (c) RENAME subcommand
  (d) DELETE subcommand
  (e) ABORT subcommand

  When more than one RENAME subcommands are specified, or when RENAME and DELETE subcommands are specified together, operation takes place in the order of specification.

Examples

`RENAME∆UN=datalist(datalst1)`

Renames unit “datalist”as “datalst1.”

`RENAME∆ED=cntl.TRUNK(P_TRUNK),ER=cntl1.REC_DATA(RECV_DATA)`

Changes export symbol “TRUNK” in unit “cntl” to “P_TRUNK.” Likewise, changes import symbol “REC_DATA” in unit “cntl1” to “RECV_DATA.”

Format

Name	Option	Subcommand	Negative Form
	None	DELETE	None

Parameters

{ UN=<unit name>
ED=<unit name >.<export symbol name> }

[, { UN=<unit name>
ED=<unit name>.<export symbol name> } ...]

Function

Specifies deletion of units or export symbols from input files.

Explanation

(1) Outline of functions:

- Deletes the specified units or export symbols from input files.
- In the case of a unit, the unit specified following “UN=” is deleted.
- In the case of an export symbol, the symbol specified following “ED=” is deleted. The export symbol name is set off by a period (.) from the name of the unit in which it exists.

(2) Restrictions in use:

- The DELETE subcommand will not affect input files already specified. This subcommand must be specified prior to specification of the input files in which the name of the unit or export symbol to be deleted is found.
- The following five subcommands can be specified immediately after the DELETE subcommand:

  (a) INPUT subcommand
  (b) EXCHANGE subcommand
  (c) DELETE subcommand
  (d) RENAME subcommand
  (e) ABORT subcommand

- When RENAME and DELETE subcommands are specified together, operation takes place in the order of specification.

Examples

```
DELETE∆UN=snap_unit
```

Deletes unit “snap_unit.”

```
DELETE∆UN=dummy,ED=main.DUMMY_ENTER
```

Deletes unit “dummy.” Also, deletes export symbol “DUMMY_ENTER” in unit “main.”

Format

Name	Option	Subcommand	Negative Form
	DEFINE	DEFINE	None
Parameter	Option UNIX:	<Import symbol name>/ { <numeric value> | [<page address>:]<address> | <export symbol name> }	
		[,<import symbol name>/ { <numeric value> | [<page address>:]<address> | <export symbol name> } ...]	
	MS-DOS:	<Import symbol name>({ <numeric value> | [<page address>:]<address> | <export symbol name> })	
		[,<import symbol name>({ <numeric value> | [<page address>:]<address> | <export symbol name> })...]	
	Sub-command	<Import symbol name> ({ <numeric value> | [<page address>:]<address> | <export symbol name> })	
		[,<Import symbol name> ({ <numeric value> | [<page address>:]<address> | <export symbol name> })...]	

Function

Specifies forced definition of import symbols.

Explanation

(1) Outline of functions:

- Forcibly defines each specified import symbol with the specified numeric value, address or export symbol value.
- Page address can be specified only for page type modules. If the page address is not specified, zero is assumed.
- Numeric values, page addresses, and addresses are specified in hexadecimal notation.

Explanation **(2) Restrictions in use:**

- When the assigned value is that of an export symbol, it must be one that has already been defined.
- If a page address is specified for non-page type modules, an error will occur and the Linkage Editor will stop execution.
- Hexadecimal numbers must start with the numbers 0 through 9.
- The range of page addresses is 0 through 0FF (hexadecimal).
- The range of addresses that can be specified varies with the H series model.

  H8/500 series: 0 through 0FFFF (hexadecimal)

  H8/300 series: 300HA: 0 through 0FFFFFF (hexadecimal)
  Others: 0 through 0FFFF (hexadecimal)

  H8S series: 2600A and 2000A: 0 through 0FFFFFF (hexadecimal)
  Others: 0 through 0FFFF (hexadecimal)

  SH series: 0 through 0FFFFFFFF (hexadecimal)

- Values defined by the DEFINE subcommand cannot be used in relocatable load modules.
- When the EXCLUDE option or subcommand is specified, non-referenced import symbols specified by the DEFINE subcommand are ignored.

Examples

```
-DEFINE=PORT10(0E8) (MS-DOS)
-DEFINE=PORT10/0E8 (UNIX)
```

Defines undefined import symbol “PORT10” as a symbol having the value 0E8 (hexadecimal).

```
DEFINE∆MAIN_RTN(PRG_EXIT)
```

Defines undefined import symbol “MAIN_RTN” as having the same value as export symbol “PRG_EXIT.”

Section 5 Input to the Linkage Editor

5.1 Object Module Files

The Linkage Editor can accept as input the object module files output by the H Series C Compiler or Assembler.

5.2 Relocatable Load Module Files

Relocatable load module files output by this Linkage Editor can be re-input. Absolute load module files cannot be re-input.

5.3 Library Files

Library files created using the H Series Librarian can be input to the Linkage Editor. Modules in library files can be specified individually, or the LIBRARY option or subcommand can be used to input modules contained in library files automatically. See further under section 4.3.3, LIBRARY—Specifies Library Files.

5.4 Default Library Files

A library file created by the H Series Librarian can be input implicitly without specifying the LIBRARY option or subcommand. This is called the default library function.

A default library is input when the following three conditions are satisfied:

- A logical name reserved as a default library name is assigned to the library file before the library files is input to the Linkage Editor.
- The NOLIBRARY option or subcommand is not specified.
- An unresolved import symbol remains after the libraries specified by the LIBRARY option or subcommand have been searched.

The Linkage Editor inputs the library files assigned to the following logical names in the order 1, 2, 3, and searches for modules that define unresolved import symbols.

1. HLNK_LIBRARY1
2. HLNK_LIBRARY2
3. HLNK_LIBRARY3

The user can specify library files corresponding to these logical names by using the setenv command for UNIX system and the SET command for MS-DOS system.

EXAMPLE:

```
set HLNK_LIBRARY1=user.lib (MS-DOS)
```

User library user.lib is assigned to the logical name HLNK_LIBRARY1.

Section 6 Output from the Linkage Editor

6.1 Linkage Lists

When the PRINT option or subcommand or the LIST subcommand is specified, the contents of a load module file being processed are output to the standard output device or to a file, as follows.

(1) Input information	(PRINT only)
(2) Link map list	(PRINT or LIST M)
(3) Export symbol list	(PRINT or LIST X)
(4) Unresolved import list	(PRINT or LIST U)
(5) RENAME/DELETE list	(PRINT only)
(6) DEFINE list	(PRINT only)

The output formats for these lists are shown below.

(1) Input Information: Information input as command line parameters, interactive mode subcommands, or subcommand files is output in the format shown in figure 6-1.

```
             H SERIES LINKAGE EDITOR Ver. 5.3
LINK COMMAND LINE
LNK -sub=func.sub
       (1)

LINK SUBCOMMANDS

     inp main
     rename ed=sin.sin0(sin1)
     delete ed=sin.sin3
     inp sin
     define undef1(100),undef2(sin1)
     print fmap
     inp cos
     inp tan                                                    (2)
     inp calc.lib(division)
     form a
     rom (SECT1, SEC1N)
     out func
     exit
     ** sin0 IS RENAMED TO sin1
     ** sin3 IS DELETED
     ** 105 UNDEFINED EXTERNAL SYMBOL (division.undef3)
```

Figure 6-1 Typical Output of Input Information

(1) Shows the character string input on the command line.

(2) Shows the character strings input as subcommands in interactive mode, or input from a subcommand file. Also shows error messages or informative messages in response to this input.

(2) Link Map List:

(a) When the PRINT option or subcommand is specified, information on each section is output in the format shown in figure 6-2.

```
        H SERIES LINKAGE EDITOR Ver. 5.3                          PAGE: 1

                      ***   LINKAGE EDITOR LINK MAP LIST   ***

SECTION NAME                 START     -  END           LENGTH
                                                   UNIT NAME      MODULE NAME

ATTRIBUTE : CODE  NOSHR  ROM
             (2)    (3)   (4)
SECT1                        H'00000000 - H'00000004    H'00000005
  (1)                                (5)       main       (6)       main
                                                (7)                  (8)
                             H'00000006 - H'00000017    H'00000012
                                                sin                 sin
                             H'00000018 - H'00000019    H'00000002
                                                cos                 cos
                             H'0000001a - H'0000002d    H'00000014
                                                tan                 tan
                             H'0000002e - H'00000043    H'00000016
                                                division            division

* TOTAL ADDRESS *            H'00000000 - H'00000043    H'00000044
                                    (9)                     (10)
```

Figure 6-2 Typical Link Map List Output Using PRINT

(b) When parameter “M” is specified in the LIST subcommand, information on each file is output in the format shown in figure 6-3.

```
      H SERIES LINKAGE EDITOR Ver. 5.3                          PAGE: 1

                      ***   LINKAGE EDITOR LINK MAP LIST   ***

FILE NAME   :   main.OBJ
                  (11)
MODULE NAME :   main
                 (8)
UNIT NAME   :   main
                 (7)

SECTION NAME          ATTRIBUTE
                      START      -  END         LENGTH

SECT1                 CODE NOSHR
  (1)                 H'00000000 - H'00000004   H'00000005
                                (5)                (6)
```

Figure 6-3 Typical Link Map List Output Using LIST

(1) Shows section names in the order in which sections are linked.

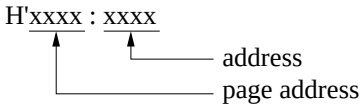
(2) Shows the attribute as follows.
DATA: data or common section
CODE: code section
DUMMY: dummy section
STACK: stack section
RESV: reserved
UNDEF: undefined
*****: unused

(3) Shows the following link attributes.
SHR: common link
NOSHR: simple link
DUMMY: dummy link
UNDEF: link attribute undefined
*****: unused

(4) Displayed for a section related to the support of storing program in ROM

ROM: ROM section (section 1 in the ROM option or subcommand)
RAM: RAM section (section 2 in the ROM option or subcommand)

(5) Shows start address and end address of the object in hexadecimal notation. In the case of page type modules, the page address and address are separated by a colon (:) as follows.



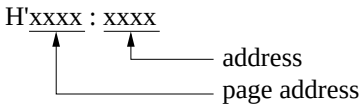
(6) Shows size of object in hexadecimal notation.

(7) Shows unit name.

(8) Shows module name.

(9) Shows start address and end address of the section.

In the case of page type modules, the page address and address are separated by a colon (:) as follows.



(10) Shows total size of the section.

(11) Shows the file name (LIST only).

(3) Export Symbols List: This list is output when there are export symbols.

(a) When the PRINT option or subcommand is specified, a list is output in the format shown in figure 6-4.

```
        H SERIES LINKAGE EDITOR Ver. 5.3                    PAGE: 1

    ***   LINKAGE EDITOR EXTERNALLY DEFINED SYMBOLS LIST   ***

SYMBOL NAME                          ADDR                  TYPE

 cos1                                H'0000000A             EQU
 sin1                                H'0000004A             DAT
 sin2                                H'0000005B             DAT
  (1)                                    (2)                 (3)
```

Figure 6-4 Typical Export Symbol List Output Using PRINT

(b) When parameter “X” is specified by the LIST subcommand, a list is output as shown in figure 6-5.

```
       H SERIES LINKAGE EDITOR Ver. 5.3                    PAGE: 1

     ***   LINKAGE EDITOR EXTERNALLY DEFINED SYMBOLS LIST   ***

SYMBOL  NAME                          ADDR                  TYPE

 cos1                                 H'0000000A             EQU
 sin1                                 H'00000000             DAT
 sin2                                 H'00000011             DAT
  (1)                                    (2)                 (3)
```

Figure 6-5 Typical Export Symbol List Output Using LIST

(1) Shows export symbols in alphabetical order.
(2) Shows the value of each export symbol in hexadecimal notation. In the case of page type modules, the page address and address are separated by a colon (:) as follows.

```
H'xxxx : xxxx
        address
        page address
```

(3) Shows the type of symbol as follows.

DAT: data/variable name

EQU: symbol name defined as constant value

ENT: entry name

***: undefined/unused

(4) Unresolved Import Symbol List: This list is output only when there are remaining undefined symbols.

(a) When the PRINT option or subcommand is specified, a list is output in the format shown in figure 6-6.

```
     H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 1

  ***   LINKAGE EDITOR UNRESOLVED EXTERNAL REFERENCE LIST   ***

FILE NAME   : calc.lib
                 (1)
MODULE NAME : division
                 (2)
UNIT NAME   : division
                 (3)

SYMBOL  NAME                      TYPE
undef3                            ***
  (4)                             (5)
```

Figure 6-6 Typical Unresolved Import Symbol List Output Using PRINT

(b) When parameter “U” is specified by the LIST command, a list is output as shown in figure 6-7.

```
      H SERIES LINKAGE EDITOR Ver. 5.3              PAGE: 1

  ***   LINKAGE EDITOR UNRESOLVED EXTERNAL REFERENCE LIST   ***

FILE NAME      : calc.lib
                   (1)
MODULE NAME    : division
                   (2)
UNIT NAME      : division
                   (3)

SYMBOL NAME                  TYPE

undef1                       ***
undef2                       ***
undef3                       ***
  (4)                         (5)
```

Figure 6-7 Typical Unresolved Import Symbol List Output Using LIST

(1) Shows name of file containing undefined symbol.

(2) Shows name of module containing undefined symbol.

(3) Shows name of unit containing undefined symbol.

(4) Shows undefined symbol names in alphabetical order.

(5) Shows undefined symbol attributes as follows.

DAT: data/variable name

ENT: entry name

***: undefined/unused

(5) RENAME/DELETE List: When the RENAME or DELETE subcommands are used to change the name of units or symbols or delete units or symbols, specification of the PRINT option or subcommand results in output of a list in the format shown in figure 6-8.

```
      H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 1

                  ***   LINKAGE EDITOR RENAME/DELETE LIST   ***

FILE NAME  : sin.OBJ
               (1)
UNIT NAME  : sin
              (2)

        FROM NAME         TO NAME                  TYPE      RENAME/DELETE

        sin0              sin1                     ED        RENAME
        sin3               (4)                     ED        DELETE
         (3)                                       (5)          (6)
```

Figure 6-8 Typical RENAME/DELETE List

(1) Shows names of files containing the unit or symbol to be renamed or deleted in the order input.

(2) Shows the unit name. If the unit was renamed or deleted, the old unit name is shown.

(3) Shows the name before changed.

(4) Shows the name after changed. No name is shown in case of a DELETE.

(5) Shows the type specified by subcommand, as follows.

UN: unit name
ED: export symbol
ER: import symbol

(6) Shows whether the subcommand was a RENAME or a DELETE.

(6) DEFINE List: When an import symbol is forcibly defined using the DEFINE option or subcommand, specification of the PRINT option or subcommand results in output of a list in the format shown in figure 6-9.

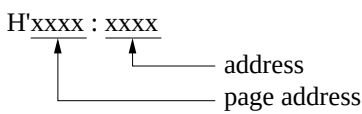
```
      H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 1

             ***   LINKAGE EDITOR DEFINE LIST   ***

UNDEFINED SYMBOL    DEFINED SYMBOL         DEFINED VALUE

undef1                                     H'00000100
undef2              sin1                   H'0000004A
  (1)                 (2)                      (3)
```

Figure 6-9 Typical DEFINE List

(1) Shows forcibly defined symbol name.

(2) Shows the name of the export symbol which is specified.

(3) Shows the value of the defined symbol in hexadecimal notation. In the case of page type modules, the page address and address are separated by a colon (:) as follows.



6.2 Load Module File

The Linkage Editor links a number of object modules or relocatable load module files and outputs them as a single load module file. Depending on the specification made with the FORM option or subcommand, the load module file is output in either absolute or relocatable format. A detailed explanation of the FORM option and subcommand is given in section 4.5.3, FORM – Specifies Output Load Module File Format.

6.3 Console Messages

The Linkage Editor shows the following messages on the standard output device.

(1) Opening Message: This is displayed when Linkage Editor command name “LNK” is input.

```
H  SERIES LINKAGE EDITOR Ver. 5.3
Copyright (C) Hitachi, Ltd. 1989
Licensed Material of Hitachi, Ltd.
```

(2) Normal Completion Message: This is displayed when the load module file editing has been completed normally.

```
LINKAGE  EDITOR  COMPLETED
```

(3) Abort Message: This is displayed when the load module file editing is ended before completion, due either to an error or to specification of an ABORT subcommand.

```
LINKAGE  EDITOR  ABORT
```

(4) Subcommand Request Prompt: In interactive mode, a colon (:) indicates that the Linkage Editor is waiting for subcommand input.

```
:
```

(5) Subcommand Continuation Prompt: When continuation of a subcommand is specified during interactive mode execution, a minus sign (–) indicates that the Linkage Editor is waiting for continuation of the input.

```
–
```

(6) Informative Message: Informative messages indicate the result of Linkage Editor processing, for example when units are replaced or when an export symbol is renamed. The messages are output in the following format.

```
** Δ <information>
↑
1st column
```

A list of informative messages is given in table 6-1. A unit name can be displayed as <External name> in table 6-1.

Table 6-1 List of Informative Messages

No.	(Informative Message) (Meaning of Message)
1	<Unit name 1> IS REPLACED WITH <unit name 2>(<file name>)
	<Unit name 1> has been replaced by <unit name 2> from <file name>.
2	<External name 1> IS RENAMED TO <external name 2>
	Name of <external name 1> has been changed to that of <external name 2>.
3	<External name> IS DELETED
	<External name> has been deleted.
4	DUPLICATE UNIT-(<unit name>) IN (<file name>) IS DELETED
	More than one units of the same name <unit name> have been found, and the unit of that name in <file name> has been deleted.
5	<Import symbol name> CANNOT DEFINE
	<Import symbol name> could not be found, and therefore could not be forcibly defined.
6	<External name> CANNOT RENAMED
	<External name> could not be found, and therefore could not be renamed.
7	<External name> CANNOT DELETED
	<External name> could not be found, and therefore could not be deleted.
8	<Unit name> CANNOT REPLACED
	<Unit name> could not be found, and therefore could not be replaced.

Section 7 Error Messages

When incorrect options or subcommands are specified, or if an error is detected during the linkage process, an error message is output. The Linkage Editor outputs error messages in the following form.

```
**Δ <Error number>Δ<error message>[(<additional information>)]
```

1st column

Error Number: The first digit indicates the level of the error (xx represent the second and third digits).

1xx: Warning	:	Processing of the particular module is skipped.
2xx: Error	:	In the case of input from the command line or a subcommand file, processing is stopped. In interactive mode, processing of the subcommand is stopped when the error is detected, and the next subcommand is requested.
3xx: Fatal error	:	Processing is stopped.

A list of errors is given below in tables 7-1, 7-2, and 7-3 in the following format.

Error Number	Error Message	Additional Information
	Nature of Error	
	Linkage Editor actions and corrective actions	

Notation used in table: —: No additional information

Table 7-1 List of Warning Messages

101	DUPLICATE OPTION/SUBCOMMAND	Option/subcommand name
	The same option or subcommand was specified more than once.	
	Only the last-specified option or subcommand is valid.	
102	IDENTIFIERCHARACTEREXCEEDS 251	Name
	Name of a unit, section, or symbol over 251 characters was specified.	
	Name is valid up to 251th character. The rest is ignored.	
104	DUPLICATE SYMBOL	Symbol name
	The same export symbol is defined more than once.	
	Only the first appearing symbol is valid.	
105	UNDEFINED EXTERNAL SYMBOL	Unit name, symbol name
	An undefined symbol was imported.	
	The import is invalid, and zero is assumed as the value.	
106	REDEFINED SYMBOL	Symbol name
	A previously defined symbol was defined using the DEFINE subcommand or option.	
	The DEFINE specification is invalid.	
107	SECTION ATTRIBUTE MISMATCH	Section name
	Two sections with the same name but different attributes or boundary alignment were input.	
	The sections are processed as separate sections.	
108*	RELOCATION SIZE OVERFLOW	Unit name, section name—offset value
	Relocation result exceeds the relocation size.	
	Result is rounded off to fit the relocation size.	
109	ENTRY POINT MULTIPLY DEFINED	—
	Execution start addresses were specified in more than one object modules.	
	The first appearing execution start address is valid.	
110	SECTION ADDRESS EXCEED PAGE BOUNDARY	Section name
	A section overlaps a page boundary.	
	Specify the AUTOPAGE option or subcommand.	
111	DUPLICATE SECTION NAME	Section name
	Same section name was specified in options or subcommands.	
	The first section is valid.	
112	ILLEGAL CPU INFORMATION FILE FORMAT	—
	The file format of the CPU information file is incorrect.	
	The CPU option or subcommand specification is invalid.	

Table 7-1 List of Warning Messages (cont)

113	CONFLICTING DEVICE TYPE	—
	The specified CPU information file is for a different CPU from that for which the input object module is intended.	
	The CPU information file specification is invalid.	
114	SECTION IS NOT IN SAME MEMORY AREA	Section name: xxxx-yyyy
	A section overlaps different memory areas. Addresses xxxx to yyyy are not allocated to one memory area.	
	The section is output to the load module without change.	
115	INACCESSIBLE ADDRESS RANGE	Section name
	A section was assigned to a memory area that cannot be used.	
	The section is output to the load module without change.	
116	INVALID CPU OPTION/SUBCOMMAND	—
	The CPU option or subcommand was specified for a relocatable load module file.	
	The CPU option or subcommand specification is invalid.	
117	ADDRESS SPACE DUPLICATE	—
	Sections overlap.	
	The load module is output as is.	
118	INVALID UDF OPTION/SUBCOMMAND	—
	The NOUDF option or subcommand was specified for an absolute output load module.	
	The NOUDF option or subcommand is invalid.	
119	RELOCATION VALUE IS ODD	Unit name, section name—offset value
	Relocation value for the displacement is odd.	
	The LSB is rounded down to fit to the relocation size.	
120	START ADDRESS NOT SPECIFIED FOR SECTION	Section name
	A section that has not been specified with the START option/subcommand was found.	
	Check the section name.	
121	CANNOT FIND SECTION	Section name
	The specified section cannot be found.	
	The section specification is ignored.	
122	TOOLONGSUBCOMMANDLINE	—
	Symbols are replaced with the corresponding directory names, and the file name exceeds 511.	
	The file name is valid up to the 511th character.	

Table 7-1 List of Warning Messages (cont)

123	TOO MANY DIRECTORY COMMANDS	—
	More than 16 directory names have been specified with the DIRECTORY subcommand.	
	Up to 16th specification is valid.	
124	NO DEBUG INFORMATION	—
	The DEBUG or SDEBUG option/subcommand has been specified for the file having no debugging information.	
	Specify the debug option at compilation or assembly.	

Note: The following describes the generating condition, generating program examples, and corrective actions for warning 108 (RELOCATIONSIZEOVERFLOW).

Warning Generating Condition: When the linkage editor determines the program addresses, if a data size designated at assembly or compilation is exceeded, warning message 108 is output.

Warning Generating Program Examples:

- H8S, H8/300 series

  Example 1

```
          .EXPORT    SYM1
SYM1      .EQU       H'1000
            .
            .
            .
```

Program 1

```
          .IMPORT    SYM1
            .
            .
            .
          MOV.B      #SYM1,R1L   (1)
```

Program 2

When the above two programs are assembled and linked, the instruction at (1) references SYM1 in byte size and therefore the referenced value must be within the range from -128 to +255. However, SYM1 is defined as H'1000 (4096) in program 1, which exceeds the range, and warning 108 is output.

Example 2

```
          .EXPORT    SYM2
SYM2      .EQU       H'C0   (2)
            .
            .
            .
```

Program 3

```
          .IMPORT    SYM2
            .
            .
            .
          MOV     @SYM2:8,R0L   (3)
```

Program 4

When the above two programs are assembled and linked, SYM 2 is referenced in 8-bit absolute addressing mode at (3). The access range in 8-bit absolute addressing is 65280 to 65535 (H'FF00 to H'FFFF). However, SYM 2 is defined as H'C0 at (2), which exceeds the range, and

warning 108 is output. In this case, @SYM2:8 accesses address H'FFC0, and therefore, when @H'FFC0 is the target address, this warning message can be ignored.

- H8/500 series

Example 3

```
        .EXPORT     SYM3
SYM3    .EQU        H'FF
          .
          .
          .
```

Program 5

```
        .IMPORT     SYM3
          .
          .
          .
        MOV   @(SYM3:8,R2),R3 (4)
```

Program 6

When the above two programs are assembled and linked, the instruction at (4) references SYM3 in 8-bit size and therefore the referenced value must be within the range from -128 to +127. However, SYM3 is defined as H'FF (255) in program 5, which exceeds the range, and warning 108 is output.

Example 4

```
        .SECTION    SEC1,CODE
SYM4        .EQU     $              ;  Sets a location value to a symbol
            .
            .
            .
        MOV         @SYM4:8,R0     ;  Transfers 2-byte data at the address pointed
                                       to by the location        (5)
```

When the above program is assembled and linked with specifying the start address of section (SEC1) as address 1000 (hexadecimal), the SYM4 value becomes H'1000, which exceeds the 1-byte data size, and warning 108 is output. In this case, when the base register (BR) is set to H'10 before the instruction at (5) is executed, this message can be ignored.

Example 5

```
        .EXPORT     SYM3
SYM3    .EQU        H'FF
          .
          .
          .
```

Program 7

```
        .IMPORT     SYM3
          .
          .
          .
        MOV   @(SYM3:8,R2),R3 (4)
```

Program 8

When the above two programs are assembled and linked, the SYM5 value referenced at (6) is defined as H'2000 in program 9, which exceeds the 1-byte data size, and warning 108 is output.

In the same way as example 4, when the base register (BR) is set to H'20 before the instruction at (6) is executed, this message can be ignored.

Corrective Actions: When the warning message cannot be ignored, take the following corrective actions.

- H8S, H8/300 series

  In example 1, the following two corrective actions can be taken:

  — Modifying the instruction operation size to word
  Modify, at (1) in program 2, MOV.B to MOV.W and R1L to R1.

  — Extracting the high-order or low-order one byte of the label (SYM1) value
  To extract the high-order byte, modify #SYM1 to #HIGH SYM1 at (1).
  To extract the low-order byte, modify #SYM1 to #LOW SYM1.

  In example 2, modify H'C0 to H'FFC0 at (2) in program 3.

- H8/500 series

  In example 3, modify SYM3:8 to SYM3:16 at (4) in program 6 when the label (SYM3) value exceeds the 1-byte data size.

  In example 4, modify @SYM4:8 to @SYM4:16 at (5) in the program.

  In example 5, modify @SYM5:8 to @SYM5:16 at (6) in program 8.

Warning Message 108 Output Format: Output in the following format:

** 108 RELOCATIONSIZEOVERFLOW (<unit name> . <section name> - <offset value>)

This message means that the data overflow has occurred <offset value> addresses after the start address of the section indicated by <unit name> . <section name>. Here, <unit name> means the file name.

Table 7-2 List of Error Messages

201	ILLEGAL SUBCOMMAND/OPTION	—
	An illegal subcommand (or option) was specified.	
	Specify a valid subcommand (or option).	
202	SYNTAX ERROR	—
	Syntax of the specified subcommand (or option) is incorrect.	
	Check the syntax and respecify the subcommand (or option).	
203	TOO LONG SUBCOMMAND LINE	—
	Length of the subcommand entry exceeds 255 characters.	
	Respecify, keeping the length within 255 characters.	
204	ILLEGAL SUBCOMMAND SEQUENCE	—
	Order of subcommand specification is invalid.	
	Check the order of subcommand specification and respecify.	
207	ILLEGAL SECTION NAME	Section name
	The specified section name is invalid.	
	Specify a proper section name.	
208	ILLEGAL SYMBOL NAME	Symbol name
	The specified symbol name is invalid.	
	Specify a proper symbol name.	
210	TOO MANY INPUT FILES	—
	Attempt was made to input more than 256 input files at one time.	
	Create a relocatable load module file, then specify the remaining input files by re-inputting the load module file.	
211	CANNOT FIND FILE	File name
	The specified file cannot be found.	
	Check the specified file name, then respecify.	
212	CANNOT FIND UNIT	Unit name
	The specified unit cannot be found.	
	Check the specified unit name, then respecify.	
213	CANNOT FIND MODULE	Module name
	The specified module cannot be found.	
	Check the specified module name, then respecify.	

Table 7-2 List of Error Messages (cont)

214	DUPLICATE START ADDRESS SPECIFIED	—
	The same start address was specified more than once.	
	Change the start address, then re-input.	
216	PAGE ADDRESS EXCEEDED	—
	A page address exceeds the permitted range.	
	Check the page address and respecify.	
217	SUBCOMMAND COMMAND IN SUBCOMMAND FILE	—
	The SUBCOMMAND subcommand appeared in a subcommand file.	
	Remove the SUBCOMMAND subcommand from the subcommand file.	
219	INVALID ADDRESS	address
	The specified address exceeds the permitted range.	
	The specified address exceeds the address range of the specified device. Check the value of the specified address, then re-execute.	
220	TOO MANY ROM COMMANDS	—
	More than 10 pairs of section names were specified in a ROM subcommand.	
	Specify 10 pairs or less.	
221	CANNOT CREATE ABSOLUTE MODULE	Module name
	An undefined import symbol was found.	
	Resolve the address for the symbol.	
222	DIVISION BY ZERO IN RELOCATION VALUE	Unit name . section name—offset
	The input object file includes a division by zero.	
	Check the relocation operation and make the object file that has no division by zero.	

Table 7-3 List of Fatal Error Messages

301	ILLEGAL COMMAND PARAMETER	—
	An illegal command parameter was specified.	
	Check the command parameters and re-execute.	
302	CANNOT OPEN FILE	File name
	The file cannot be opened.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
303	CANNOT READ INPUT FILE	File name
	The file cannot be input.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
304	CANNOT WRITE OUTPUT FILE	File name
	The file cannot be output.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
305	CANNOT CLOSE FILE	File name
	The file cannot be closed.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
306	ILLEGAL FILE FORMAT	File name
	The specified file format is incorrect.	
	Check the file contents and specified file name, then re-execute. This message is output when the object file format is illegal, for example because there are two or more import symbols with the same name in the same unit, or two external symbol names were made identical by the RENAME subcommand.	
307	ILLEGAL RECORD FORMAT	File name
	There is an illegal record in the specified file, or division by zero occurred.	
	Check the source program contents. Re-assemble or recompile, then re-execute.	
308	SECTION ADDRESS OVERFLOW	Section name of the specified device
	The address allocated to a section exceeds the allowable range.	
	The address allocated to the section exceeds the address range of the specified device. Change the section start address or rearrange the user program, then re-execute.	

Table 7-3 List of Fatal Error Messages (cont)

309	ADDRESS OVERFLOW	—
	The specified address exceeds the address range allowed for the particular CPU.	
	Check the specified address, then re-execute.	
310	MEMORY OVERFLOW	—
	There is no space remaining in the Linkage Editor's usable memory.	
	Expand the memory or revise the user program, then re-execute.	
311	PROGRAM ERROR	nnn
	There is an error in the Linkage Editor program.	
	The Linkage Editor is inoperable. Check the program error number (nnn), then contact your Hitachi representative.	
312	ILLEGAL START ADDRESS ALIGNMENT	Address
	The specified address conflicts with the boundary alignment number of the object module.	
	Check the boundary alignment number of the object module, then re-execute.	
314	CANNOT FIND SECTION	Section name
	The specified section name cannot be found.	
	Check the section name, then respecify.	
319	AUTOPAGE SPECIFIED AT NON-PAGE TYPE	—
	The AUTOPAGE option/subcommand was specified when non-page type files were input.	
	Check the input file contents, then respecify.	
321	PAGE ADDRESS OVERFLOW	—
	The page address overflows the allowable range.	
	Change the section start address or the user program so that the page address will be within the allowable range of 0 - 0FF (hexadecimal), then re-execute.	
322	PAGE ADDRESS SPECIFIED AT NON–PAGE TYPE	—
	For a non-page type input file, a page address was specified with the START or DEFINE option/subcommand.	
	Check the specified file name and option or subcommand content, then re-execute.	
323	SECTION SPECIFIED AT ROM OPTION/ SUBCOMMAND DOES NOT EXIST	Section name
	A section specified in a ROM command does not exist.	
	Check the section name, and respecify.	

Table 7-3 List of Fatal Error Messages (cont)

325	ILLEGAL START SECTION	Section name
	A section specified by a START command has an illegal attribute.	
	Check the section attributes, and respecify.	
326	CANNOT READ	—
	Input failed from a file (including the standard input device).	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
327	SYMBOL ADDRESS OVERFLOW	Symbol name
	The address assigned to a symbol exceeded the permitted range for the specified device.	
	Change the section start address or rearrange the user program, then re-execute.	
328	ILLEGAL ROM SECTION	Section name
	Section 2 specified in a ROM subcommand or option is invalid.	
	The size of section 2 is not 0, section 2 is the absolute section or the attribute of section 2 is different from that of section 1. Check the size and attribute of section 2, and respecify.	
329	INVALID MEMORY MAP	—
	Memory allocation does not match the one specified in the CPU information file, or it overlaps different types of memory.	
	Check the CPU information file and the input files.	
330	ILLEGAL FILE FORMAT (INPUT ABSOLUTE FILE)	—
	An absolute load module was input.	
	Check the input files and respecify them.	
331	ILLEGAL FILE FORMAT (MISMATCH OBJECT FORMAT VERSION)	—
	The input files have different object formats.	
	Check the input files and respecify them.	
332	ILLEGAL FILE FORMAT (INPUT MISMATCH CPU TYPE)	—
	The input files are not for the H series or SH series.	
	Check the input files and respecify them.	

Section 8 Restrictions

Restrictions on the Linkage Editor are shown in table 8-1. If the numerical restrictions are exceeded, linkage operations cannot be performed.

Table 8-1 Restrictions on Linkage Editor Processing

No.	Item	Restrictions	Remarks
1	Number of input files	256 max.	
2	Input file formats	• Object module file output by assembler or compiler. • Relocatable load module file. • Library file created using Librarian.	
3	Address/notation	Hexadecimal only. The range depends on the H series type.	H8/500 series: 0-0FFFF H8/300 series: • 300HA: 0-0FFFFFF • Others: 0-0FFFF H8S series: • 2600A and 2000A: 0-0FFFFFFFF • Others: 0-0FFFF SH series: 0-0FFFFFFFF
4	Names of modules, units, sections, symbols	Up to 251 characters.	
5	Number of modules, units, sections, export symbols, import symbols	65,535.	Assumes no prior restrictions on memory of system on which Linkage Editor is executed.

Appendix A Example of Use of Linkage Editor

In this sample application, the 11 object modules and one library file shown in table A-1 are input into the Linkage Editor.

Table A-1 List of Input Files

No.	File Name	Type of File
1	main.obj	Object module file
2	init.obj	
3	cmndanl.obj	
4	cmndprc.obj	
5	table.obj	
6	term.obj	
7	keyin.obj	
8	file.obj	
9	printer.obj	
10	display.obj	
11	commu.obj	
12	function.lib	Library file

Library file “function.lib” consists of the 14 modules listed in table A-2.

Table A-2 List of Modules in Library File

No.	Module Name
1	mvdata
2	upshft
3	comp
4	expr
5	rmargin
6	lmargin
7	sum
8	number
9	zerosprs
10	ascbin
11	binasc
12	cnvbcd
13	portio
14	dos

Linkage Execution: Input the following command to execute module linkage. In this example, subcommands are input from subcommand file “exlink.sub,” and execution is controlled by these subcommands.

```
lnk∆-SUBCOMMAND=exlink.sub (RET)
```

The contents of subcommand file “exlink.sub” are shown in figure A-1.

```
;
;  First Linkage Process
;
form      r               ; Relocatable Load Module
input     main            ; Input "main.obj"
input     init            ; Input "init.obj"
input     cmndanl         ; Input "cmndanl.obj"
input     cmndprc         ; Input "cmndprc.obj"
input     table           ; Input "table.obj"
input     term            ; Input "term.obj"
library   function        ; Library "function.lib"
output    program1        ; Output "program1.rel"
print     program1        ; Print "program1.map"
end
;
;  Second Linkage Process
;
input     program1.rel    ; Input "program1.rel"
input     keyin           ; Input "keyin.obj"
input     file            ; Input "file.obj"
input     printer         ; Input "printer.obj"
input     display         ; Input "display.obj"
input     commu           ; Input "commu.obj"
library   function        ; Library "function.lib"
                          ; Sequence of Sections
start     program1,program2,function,global,local,f_local,stack_area
output    example         ; Output "example.abs"
print     example         ; Print "example.map"
exit
```

Figure A-1 Subcommand File “exlink.sub”

As figure A-1 shows, two linkage processes are carried out, using the multilinkage function. In the first linkage process, six object module files and the library file are input, and relocatable load module file “program1.rel” and linkage list “program1.map” are output. In the second linkage process, load module file “program1.rel” is re-input, and the remaining object module files are input. Absolute load module file “example.abs” and linkage list “example.map” are output.

Linkage list, “program1.map” output in the first linkage process is shown in figure A-2. Linkage list “example.map” output in the second linkage process is shown in figure A-3.

```
     H SERIES LINKAGE EDITOR Ver. 5.3

LINK COMMAND LINE

lnk -subcommand=exlink.sub

LINK SUBCOMMANDS

;
;  First Linkage Process
;
form       r             ; Relocatable Load Module
input      main          ; Input "main.obj"
input      init          ; Input "init.obj"
input      cmndanl       ; Input "cmndanl.obj"
input      cmndprc       ; Input "cmndprc.obj"
input      table         ; Input "table.obj"
input      term          ; Input "term.obj"
library    function      ; Library "function.lib"
output     program1      ; Output "program1.rel"
print      program1      ; Print "program1.map"
end
** 105 UNDEFINED EXTERNAL SYMBOL(main.keyin)
** 105 UNDEFINED EXTERNAL SYMBOL(cmndprc.printer)
** 105 UNDEFINED EXTERNAL SYMBOL(cmndprc.file)
** 105 UNDEFINED EXTERNAL SYMBOL(cmndprc.keyin)
** 105 UNDEFINED EXTERNAL SYMBOL(cmndprc.commu)
** 105 UNDEFINED EXTERNAL SYMBOL(cmndprc.display)
** 105 UNDEFINED EXTERNAL SYMBOL(term.file)
```

Figure A-2 Linkage List “program1.map” (Input Information)

```
             H SERIES LINKAGE EDITOR Ver. 5.3                        PAGE: 1

                    ***   LINKAGE EDITOR LINK MAP LIST   ***

SECTION NAME       START      - END         LENGTH
                                   UNIT NAME    MODULE NAME
ATTRIBUTE : CODE NOSHR

program1           H'00000000 - H'00000349  H'0000034a
                                     main            main
                   H'0000034a - H'00000467  H'0000011e
                                     init            initialize
                   H'00000468 - H'0000055d  H'000000f6
                                     cmndanl         command_analize
                   H'0000055e - H'000007e7  H'0000028a
                                     cmndprc         command_process
                   H'000007e8 - H'0000091f  H'00000138
                                     term            terminate
* TOTAL ADDRESS *  H'00000000 - H'0000091f  H'00000920

ATTRIBUTE : DATA NOSHR

local              H'00000000 - H'00001ELF  H'00001e20
                                     main         main
                   H'00001e20 - H'00001e3f  H'00000020
                                     init         initialize
                   H'00001e40 - H'00003c7f  H'00001e40
                                     cmndanl      command_analize
                   H'00003c80 - H'000222bf  H'0001e640
                                     cmndprc      command_process
                   H'000222c0 - H'000222df  H'00000020
                                     term         terminate
* TOTAL ADDRESS *  H'00000000 - H'000222df  H'000222e0

ATTRIBUTE : DATA NOSHR

global             H'00000000 - H'000015cf  H'000015d0
                                     table        global_table
* TOTAL ADDRESS *  H'00000000 - H'000015cf  H'000015d0

ATTRIBUTE : STACK NOSHR

stack_area         H'00000000 - H'001e1fff  H'001e2000
                                     table        global_table
* TOTAL ADDRESS *  H'00000000 - H'001e1fff  H'001e2000
```

Figure A-2 Linkage List “program1.map” (Link Map List)

```
                  H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 2

                     ***   LINKAGE EDITOR LINK MAP LIST   ***

SECTION NAME      START      - END            LENGTH
                                            UNIT NAME     MODULE NAME

ATTRIBUTE : CODE NOSHR

function          H'00000000 - H'0000001b     H'0000001c
                                   comp                   compare_string
                  H'0000001c - H'0000010f     H'000000f4
                                   expr                   expression
                  H'00000110 - H'00000163     H'00000054
                                   mvdata                 move_data_string
                  H'00000164 - H'00000193     H'00000030
                                  upshft                  upshift_character
* TOTAL ADDRESS * H'00000000 - H'00000193     H'00000194

ATTRIBUTE : DATA NOSHR

f_local           H'00000000 - H'0000000b     H'0000000c
                                  comp                    compare_string
                  H'0000000c - H'0000011b     H'00000110
                                  expr                    expression
                  H'0000011c - H'0000011f     H'00000004
                                  upshft                  upshift_character
* TOTAL ADDRESS * H'00000000 - H'0000011f     H'00000120
```

Figure A-2 Linkage List “program1.map” (Link Map List) (cont)

```
H SERIES LINKAGE EDITOR Ver. 5.3                                PAGE: 1

***   LINKAGE EDITOR EXTERNALLY DEFINED SYMBOLS LIST   ***

  SYMBOL NAME              ADDR              TYPE

  cmndanl                  H'00000000        DAT
  cmndprc                  H'00000000        DAT
  cmndtbl                  H'000000C8        DAT
  comp                     H'00000000        DAT
  expr                     H'00000000        DAT
  fltbl                    H'000003C8        DAT
  header                   H'00000000        DAT
  init                     H'00000000        DAT
  keybuf                   H'000001C8        DAT
  main                     H'00000000        DAT
  mvdata                   H'00000000        DAT
  prbuf                    H'000014C8        DAT
  recbuf                   H'000013C8        DAT
  stackarea                H'00000000        DAT
  term                     H'00000000        DAT
  upshft                   H'00000000        DAT
```

Figure A-2 Linkage List “program1.map” (Export Symbol List)

```
            H SERIES LINKAGE EDITOR Ver. 5.3                        PAGE: 1

            *** LINKAGE EDITOR UNRESOLVED EXTERNAL REFERENCE LIST ***

FILE NAME       : main.obj

MODULE NAME     : main
UNIT NAME       : main

                       SYMBOL NAME                       TYPE

                       keyin                              ***

FILE NAME       : cmndprc.obj

MODULE NAME     : command_process
UNIT NAME       : cmndprc

                       SYMBOL NAME                       TYPE

                       commu                              ***
                       display                            ***
                       file                               ***
                       keyin                              ***
                       printer                            ***

FILE NAME       : term.obj

MODULE NAME     : terminate
UNIT NAME       : term

                   SYMBOL NAME                       TYPE

                   file                               ***
```

Figure A-2 Linkage List “program1.map” (Undefined Symbol List)

```
             H SERIES LINKAGE EDITOR Ver. 5.3
LINK COMMAND LINE

LINK SUBCOMMANDS

;
; Second Linkage Process
;
input        program1.rel    ; Input "program1.rel"
input        keyin           ; Input "keyin.obj"
input        file            ; Input "file.obj"
input        printer         ; Input "printer.obj"
input        display         ; Input "display.obj"
input        commu           ; Input "commu.obj"
library      function        ; Library "function.lib"
                             ; Sequence of Sections
start        program1,program2,function,global,local,f_local,stack_area
output       example         ; Output "example.abs"
print        example         ; Print "example.map"
exit
```

Figure A-3 Linkage List “example.map” (Input Information)

```
                    H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 1

                        ***   LINKAGE EDITOR LINK MAP LIST ***

SECTION NAME        START      - END              LENGTH
                                                 UNIT NAME    MODULE NAME
ATTRIBUTE : CODE NOSHR
program1            H'00000000 - H'00000349    H'0000034a
                                             main             program1
                    H'0000034a - H'00000467    H'0000011e
                                             init             program1
                    H'00000468 - H'0000055d    H'000000f6
                                             cmndanl          program1
                    H'0000055e - H'000007e7    H'0000028a
                                             cmndprc          program1
                    H'000007e8 - H'0000091f    H'00000138
                                             term             program1
* TOTAL ADDRESS *   H'00000000 - H'0000091f    H'00000920

ATTRIBUTE : CODE NOSHR
program2            H'00000920 - H'00000b1f    H'00000200
                                             keyin            input_keyboard
                    H'00000b20 - H'00000c47    H'00000128
                                             file             file_io
                    H'00000c48 - H'00000d49    H'00000102
                                             printer          output_printer
                    H'00000d4a - H'00000e61    H'00000118
                                             display          display_console
                    H'00000e62 - H'00001127    H'000002c6
                                             commu            communication
* TOTAL ADDRESS *   H'00000920 - H'00001127    H'00000808

ATTRIBUTE : CODE NOSHR
function            H'00001128 - H'00001143    H'0000001c
                                             comp             program1
                    H'00001144 - H'00001237    H'000000f4
                                             expr             program1
                    H'00001238 - H'0000128b    H'00000054
                                             mvdata           program1
                    H'0000128c - H'000012bb    H'00000030
                                             upshft           program1
                    H'000012bc - H'00001343    H'00000088
                                             lmargin          left_margin
                    H'00001344 - H'00001373    H'00000030
                                             number           numbering_items
                    H'00001374 - H'000013f3    H'00000080
                                             rmargin          right_margin
```

Figure A-3 Linkage List “example.map” (Link Map List)

```
                    H SERIES LINKAGE EDITOR Ver. 5.3                  PAGE: 2

                     ***   LINKAGE EDITOR LINK MAP LIST   ***

SECTION NAME      START      - END              LENGTH
                                               UNIT NAME   MODULE NAME
ATTRIBUTE : CODE NOSHR
function           H'000013f4 - H'0000140b   H'00000018
                                            sum           sum_items
                   H'0000140c - H'000014c7   H'000000bc
                                            zerosprs      zero_suppress
                   H'000014c8 - H'00001533   H'0000006c
                                            ascbin        ascii_to_binary
                   H'00001534 - H'00001573   H'00000040
                                            binasc        binary_to_ascii
                   H'00001574 - H'0000163f   H'000000cc
                                            cnvbcd        convert_to_bcd
                   H'00001640 - H'00001647   H'00000008
                                            dos           interface_of_dos
                   H'00001648 - H'00001657   H'00000010
                                            portio        interface_of_port
* TOTAL ADDRESS *  H'00001128 - H'00001657   H'00000530

ATTRIBUTE : DATA NOSHR
global             H'00001658 - H'00002c27   H'000015d0
                                            table         program1
* TOTAL ADDRESS *  H'00001658 - H'00002c27   H'00015d0

ATTRIBUTE : DATA NOSHR
local              H'00002c28 - H'00004a47   H'00001e20
                                            main          program1
                   H'00004a48 - H'00004a67   H'00000020
                                            init          program1
                   H'00004a68 - H'000068a7   H'00001e40
                                            cmndanl       program1
                   H'000068a8 - H'00024ee7   H'0001e640
                                            cmndprc       program1
                   H'00024ee8 - H'00024f07   H'00000020
                                            term          program1
                   H'00024f08 - H'00025127   H'00000220
                                            keyin         input_keyboard
                   H'00025128 - H'00025307   H'000001e0
                                            file          file_io
                   H'00025308 - H'0002544b   H'00000144
                                            printer       output_printer
                   H'0002544c - H'0002554f   H'00000104
                                            display       display_console
```

Figure A-3 Linkage List “example.map” (Link Map List) (cont)

```
                    H SERIES LINKAGE EDITOR Ver. 5.3                PAGE: 3

                       ***   LINKAGE EDITOR LINK MAP LIST   ***

SECTION NAME        START      - END            LENGTH
                                              UNIT NAME      MODULE NAME

ATTRIBUTE : DATA NOSHR
local               H'00025550 - H'00025713    H'000001c4
                                             commu            communication
* TOTAL ADDRESS *   H'00002c28 - H'00025713    H'00022aec

ATTRIBUTE : DATA NOSHR
f_local             H'00025714 - H'0002571f    H'0000000c
                                             comp             program1
                    H'00025720 - H'0002582f    H'00000110
                                             expr             program1
                    H'00025830 - H'00025833    H'00000004
                                             upshft           program1
                    H'00025834 - H'00025843    H'00000010
                                             lmargin          left_margin
                    H'00025844 - H'00025847    H'00000004
                                             number           numbering_items
                    H'00025848 - H'00025857    H'00000010
                                             rmargin          right_margin
                    H'00025858 - H'0002587b    H'00000024
                                             zerosprs         zero_suppress
                    H'0002587c - H'00025883    H'00000008
                                             ascbin           ascii_to_binary
                    H'00025884 - H'00025887    H'00000004
                                             binasc           binary_to_ascii
                    H'00025888 - H'000258cf    H'00000048
                                             cnvbcd           convert_to_bcd
* TOTAL ADDRESS *   H'00025714 - H'000258cf    H'000001bc

ATTRIBUTE: STACK NOSHR
stack_area          H'000258d0 - H'002078cf    H'001e2000
                                             table            program1
* TOTAL ADDRESS *   H'000258d0 - H'002078cf    H'001e2000
```

Figure A-3 Linkage List “example.map” (Link Map List) (cont)

```
H SERIES LINKAGE EDITOR Ver. 5.3                                PAGE: 1

***   LINKAGE EDITOR EXTERNALLY DEFINED SYMBOLS LIST   ***

SYMBOL NAME                ADDR                  TYPE

ascbin                     H'000014c8            DAT
binasc                     H'00001534            DAT
cmndanl                    H'00000468            DAT
cmndprc                    H'0000055e            DAT
cmndtbl                    H'00001720            DAT
cnvbcd                     H'00001574            DAT
commu                      H'00000e62            DAT
comp                       H'00001128            DAT
display                    H'00000d4a            DAT
dos                        H'00001640            DAT
expr                       H'00001144            DAT
file                       H'00000b20            DAT
fltbl                      H'00001a20            DAT
header                     H'00001658            DAT
init                       H'0000034a            DAT
keybuf                     H'00001820            DAT
keyin                      H'00000920            DAT
lmargin                    H'000012bc            DAT
main                       H'00000000            DAT
mvdata                     H'00001238            DAT
number                     H'00001344            DAT
portio                     H'00001648            DAT
prbuf                      H'00002b20            DAT
printer                    H'00000c48            DAT
recbuf                     H'00002a20            DAT
rmargin                    H'00001374            DAT
stackarea                  H'000258d0            DAT
sum                        H'000013f4            DAT
term                       H'000007e8            DAT
upshft                     H'0000128c            DAT
zerosprs                   H'0000140c            DAT
```

Figure A-3 Linkage List “example.map” (Export Symbol List)

Appendix B File Name Specifications

File names are specified in the following format:

path name	main file name	file type
(1)	(2)	(3)

(1) Path name

Specify the directory path of the directory containing the file, using slashes (/) in UNIX or back-slashes (\) in MS-DOS to delimit directory names. The default value is the current directory.

(2) Main file name

Specify the name of the file.

(3) File type

Specify the type of file separated from the main file name by a period (.).

The general rules of file naming for the Linkage Editor conform to the operating-system (OS) rules.

Example 1 (MS-DOS): \usr \tool\ prog .typ

- .typ — File type
- prog — Main file name
- \usr \tool\ — Path name

Example 2 (UNIX): /usr /tool/ prog .typ

- .typ — File type
- prog — Main file name
- /usr /tool/ — Path name

Note: If the same name is specified for the input file and output file, the input file contents will be lost. Do not use the same name for the input and output files.

Part II

Librarian Guide

Section 1 Overview

A program is usually developed by dividing it into functional modules and creating a separate source program for each module. Next, each source program module is compiled or assembled to create an object module. The object modules are then linked together using a linkage editor, resulting in an executable program.

The H Series Librarian introduced in this manual (hereafter called the Librarian) plays a vital role in this process. It brings together the many object modules output by the C compiler and assembler, as well as relocatable load modules output by the linkage editor, to make library files.

The Librarian provides the following advantages.

Simplified Module Management: The many modules making up a program (including relocatable load modules as well as object modules) are stored in a library file for the particular program. They can then be dealt with all at once. Moreover, it is possible to create generic library files that can be used later to streamline the creation of other programs.

A library file can be edited by adding, deleting, or replacing individual modules. In this way the modules can be kept up to date.

Enhanced Linkage: The Linkage Editor can search library files to find, extract, and link modules that define unresolved import symbols. Use of the library files thus makes linkage editing more efficient.

Section 2 Librarian Functions

2.1 Creating Library Files

This function makes it possible to create new library files, and to enter object modules output by the C compiler or assembler as well as relocatable load modules output by the linkage editor.

Figure 2-1 is an illustration of the library file creation concept.

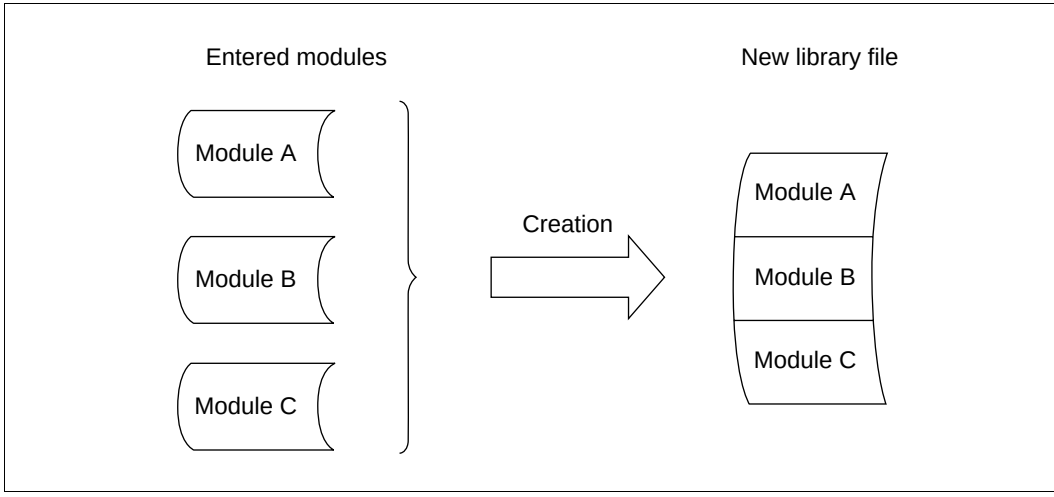


Figure 2-1 Creating a New Library File

2.2 Editing Existing Library Files

Modules can be added to, deleted from, or replaced in existing library files.

Adding Modules: Modules can be added to already existing library files. The concept of module addition is illustrated in figure 2-2.

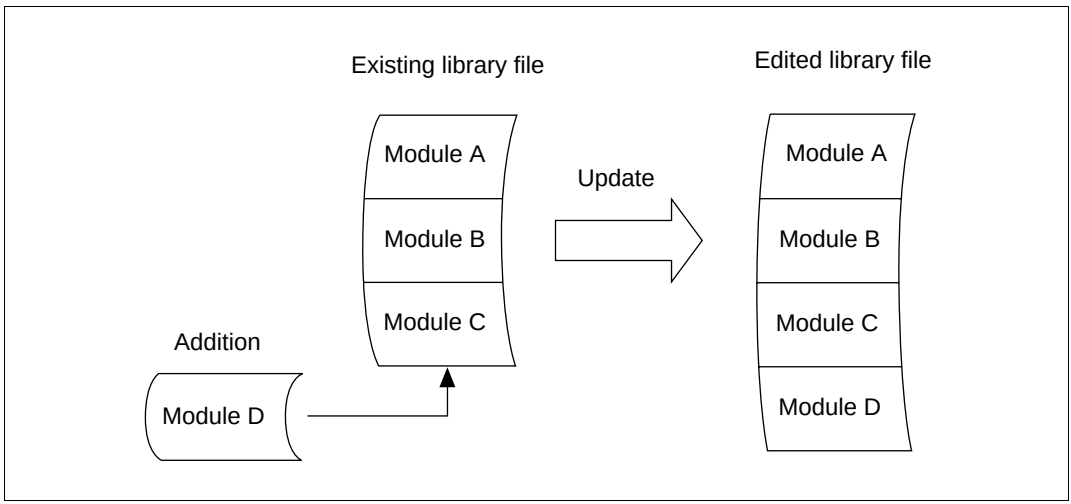


Figure 2-2 Adding a Module

Deleting Modules: Unnecessary modules can be deleted from existing library files. Figure 2-3 illustrates the module deletion concept.

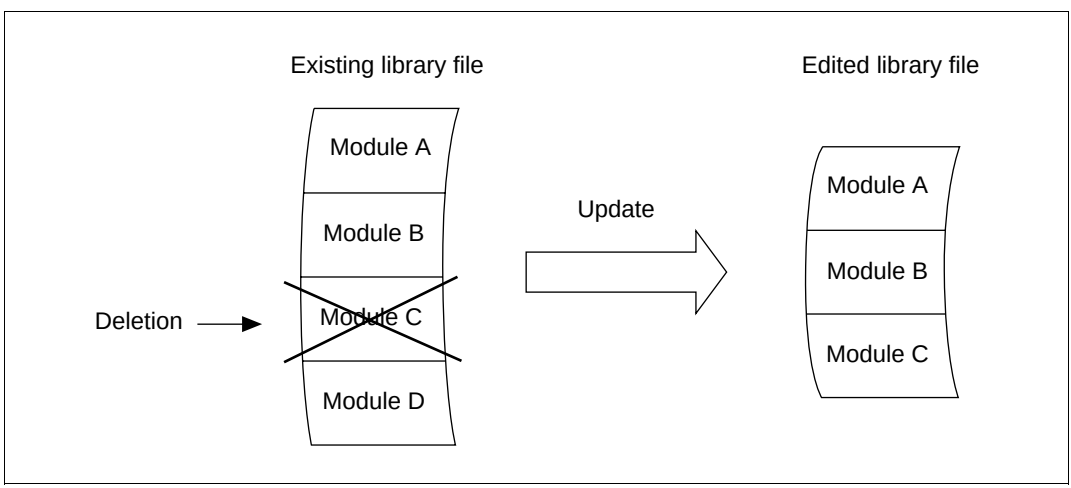


Figure 2-3 Deleting a Module

Replacing Modules: Modules in existing library files can be replaced with new modules. The concept of module replacement is illustrated in figure 2-4.

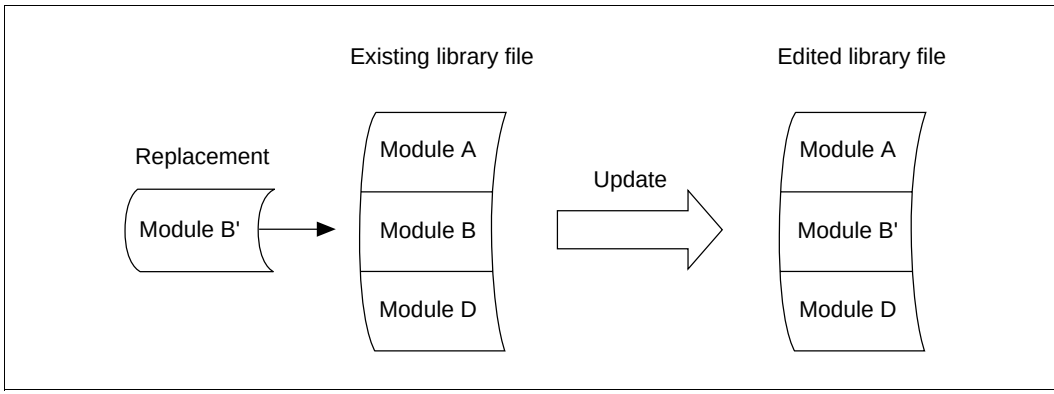


Figure 2-4 Replacing a Module

2.3 Extracting Modules from a Library File

Modules can be extracted from existing library files and used to create new library files. The concept of module extraction is illustrated in figure 2-5.

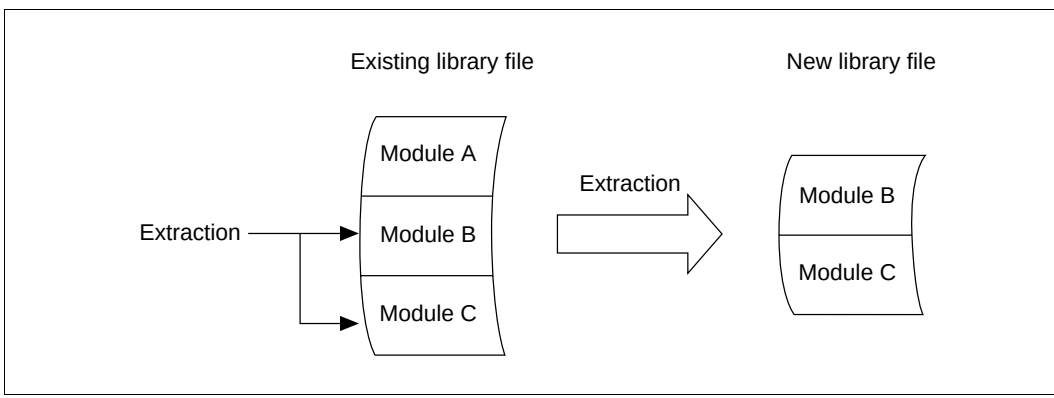


Figure 2-5 Extracting Modules

2.4 Displaying the Contents of a Library File

A librarian list giving information about the modules and export symbols in a library file can be output to a standard output device or a list file. A librarian list tells when the library file was created and when it was last revised, indicates when each module was stored, and gives the names of export symbols and other useful information.

For further details, see section 6.2, Librarian Lists.

Section 3 Executing the Librarian

To execute the Librarian, start the Librarian by entering a command line. The command line specifies the name of the library file to be edited and various options, which give instructions to the Librarian. If these instructions are sufficient, the Librarian can be executed using the command line alone. If further instructions are needed, they can be given in subcommands.

Command Line Execution: The Librarian can be executed simply by specifying a library file and options on the command line. The method is useful when library editing is relatively straightforward.

Subcommand Execution: The Librarian can also be executed by entering both a command line and subcommands. The subcommands specify input and output files and parameters that control the Librarian. This method is useful for specifying a large number of files or modules, or for editing two or more library files together. Subcommands can be entered interactively, or from a subcommand file. Details are given in section 3.3, Executing by Subcommands.

File names used on the command line and in the subcommands are specified in the following format:

path name	main file name	file type
(1)	(2)	(3)

(1) Path name

Specify the directory path of the directory containing the file, using slashes (/) in UNIX system and back-slashes (\) in MS-DOS system to delimit directory names. The default value is the current directory.

(2) Main file name

Specify the name of the file.

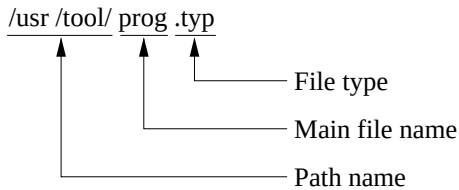
(3) File type

Specify the type of file separated from the main file name by a period (.). If omitted, the implicit type is used.

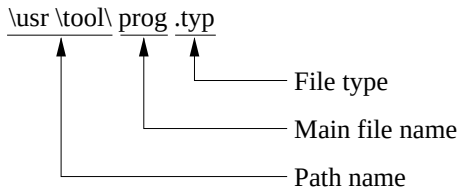
The general rules of file naming for the Librarian conform to the operating-system (OS) rules.

Note: The OS shell (command interpreter) checks the command line before passing control to the Librarian. Use characters that the OS allows on the command line.

Example (UNIX):



Example (MS-DOS):



3.1 Command Line Format

The following format is used for the Librarian command line.

lbr[Δ[<library file name>][[Δ]-<option name>[[Δ]-<option name>...]]] (RET)

- Command name: "lbr" is the command that starts the Librarian.
- Library file name: To edit or extract modules from an existing library file, type the name of the library file in the command line.
- Option names: Each option name must start with a hyphen (-). One or more spaces or tabs can also be used to separate an option name from a preceding option name or library file name, but these spaces or tabs are not required. Option names are described in detail in section 4, Librarian Options and Subcommands. The Librarian edits the library file according to the order in which the options are specified.

Specifying the Mode of Execution: The content of the command line determines whether the Librarian will be executed by the command line specifications only, or by subcommands. See table 3-1.

Table 3-1 How Command Line Specification Determines the Form of Execution

Library File Name Specification	Option Specification			
	No Option Specified	SUBCOMMAND*[1] Option Specified	CREATE Option*[1] Specified	Option Other than CREATE or SUBCOMMAND Specified
Library file name specified	—*[2]	—	—	Executed by specifying command line
No library file name specified	Executed by specifying subcommands	Executed by specifying subcommands	Executed by specifying command line	—

Notes: 1. For SUBCOMMAND and CREATE options, see section 4, Librarian Options and Subcommands.

2. The combinations of option and library file names indicated by dashes (—) are not permitted. An error will occur, and the librarian will not be executed.

3.2 Executing by Command Line

In command line execution, the Librarian is executed according to the information specified in the command line alone. Editing procedures and other conditions are specified to the Librarian in the form of options. When the editing process is straightforward and simple, command line specification is sufficient for creating or updating a library. Examples of execution by command line are given below.

Example 1:

```
lbr∆-CREATE-syslib.lib-ADD=obj00.obj,prg.lib (RET)
```

(1) (2)

(1) Creates a new library file named syslib.lib.

(2) Adds the modules in object module file obj00.obj and library file prg.lib to syslib.lib.

The CREATE option by itself will not create a library file unless modules are added using the ADD option.

Example 2:

```
lbr∆ syslib.lib-ADD=obj00.obj-DELETE=modl (RET)
```

(1) (2) (3)

(1) Designates library file syslib.lib as the file to be edited.

(2) Adds the module in object module file obj00.obj to syslib.lib.

(3) Deletes existing module mod1 from syslib.lib.

3.3 Executing by Subcommands

Since the number of characters that can be typed on the command line is limited, the command line may not be able to accommodate a large number of specifications. In such cases, subcommands are used to execute the Librarian. Subcommands can be input interactively, one at a time, from the keyboard or other standard input device. Alternatively, a subcommand file consisting of a group of subcommands can be created in advance, and subcommands can be input from this subcommand file.

3.3.1 Executing in Interactive Mode

When no library file is specified in the command line and there are no option specifications, execution proceeds in interactive mode. A colon (:) appears on the screen as a prompt, indicating that the Librarian is waiting for a subcommand to be input. In this way you can enter the necessary subcommands. This method is useful when the number of subcommands is relatively small, or when you want to check Librarian lists as you enter the subcommands.

An example of execution by interactive input of subcommands is given below. Functions of the subcommands listed here are detailed in section 4, Librarian Options and Subcommands.

Example:

```
  lbr (RET)                                        ...(1)
: CREATE∆prg.lib (RET)                             ...(2)
: ADD∆main.obj (RET)                               ...(3)
: ADD∆send.obj,receive.obj,exchange.obj (RET)      ...(4)
: ADD∆account.obj (RET)                            ...(5)
: LIST∆(S) (RET)                                   ...(6)
: EXIT (RET)                                       ...(7)
```

(1) Starts the Librarian in interactive mode.

(2) Creates a new library file named prg.lib.

(3) Adds the module in main.obj to prg.lib.

(4) Adds the modules in send.obj, receive.obj and exchange.obj to prg.lib.

(5) Adds the module in account.obj to prg.lib.

(6) Outputs a librarian list, including symbol information, to the standard output device.

(7) Terminates the Librarian operation.

3.3.2 Executing from a Subcommand File

This method uses a subcommand file that was created in advance and that contains the subcommands necessary for Librarian operations. This subcommand file is then specified on the command line as a parameter of the SUBCOMMAND option. This method is useful when many subcommands must be specified, or when the same editing process is carried out repeatedly. It eliminates the need to input subcommands from the keyboard or other standard input device each time.

Use an editor to create the subcommand file. An example of execution from a subcommand file is given below. Functions of the subcommands listed here are detailed in section 4, Librarian Options and Subcommands.

```
lbr∆-SUBCOMMAND=prglib.sub (RET)    ... (1)
```

Contents of subcommand file prglib.sub:

```
CREATE∆function.lib                 ... (2)
ADD∆sin.obj,cos.obj,tan.obj         ... (3)
ADD∆asin.obj,acos.obj,atan.obj      ... (4)
ADD∆hsin.obj,hcos.obj,htan.obj      ... (5)
ADD∆log.obj,log10.obj               ... (6)
EXIT                                ... (7)
```

(1) Starts the Librarian and inputs subcommands from subcommand file prglib.sub.

(2) Creates a new library file function.lib.

(3) Adds the modules in object module files sin.obj, cos.obj and tan.obj to function.lib.

(4) Adds the modules in object module files asin.obj, acos.obj and atan.obj to function.lib.

(5) Adds the modules in object module files hsin.obj, hcos.obj and htan.obj to function.lib.

(6) Adds the modules in object module files log.obj and log10.obj to function.lib.

(7) Terminates Librarian operations.

3.4 Terminating Librarian Operations

When the Librarian terminates operations, it gives the system a return code indicating an error level. The return code can be used to control the execution of a command file. The error code has the values shown in table 3-2, depending on the error level.

Table 3-2 Return Code Depending on Error Level

	Return Code	
Error level	**UNIX**	**MS-DOS**
Normal termination	0	0
Warning	0	0
Error	1	2
Fatal error	1	4

Section 4 Librarian Options and Subcommands

Options and subcommands tell the Librarian what editing operations to perform. The three main functions of options and subcommands are file control, execution control, and list display. These functions can be used individually or in combination to create and edit library files.

Options and subcommands have the same names and equivalent functions, but are specified in different formats. Moreover, there are some specifications which can be made only with options, and others only with subcommands. Sections 4.1, Option and Subcommand Formats, and 4.2, List of Options and Subcommands, must accordingly be read carefully. Option and subcommand functions are outlined below.

File Control Functions: File control functions indicate the name of the library file to be edited, or the name of a library file to which extracted modules are to be output.

Execution Control Functions: Execution control functions instruct the Librarian to perform editing operations, or terminate its processing. These functions are used, for example, to input subcommands from a subcommand file, to create a new library file, or to update a library file.

List Display Functions: List display functions are used to display information such as names of modules stored in a library file, or export symbol names.

4.1 Option and Subcommand Formats

Option and Subcommand Structure:

(a) Name

The name gives the name of the option or subcommand. For the names, refer to section 4.2, List of Options and Subcommands.

(b) Parameters

The parameters give the names of files,*[1] module,*[2] etc. on which the option or subcommand operates. There are different requirements and methods of specification depending on the type of option or subcommand. For details, refer to section 4.3, File Control, section 4.4, Execution Control, and section 4.5, List Display.

Options and subcommands differ as to the way of separating the name from the parameters. Options use an equal sign (=), while subcommands use one or more spaces or tabs.

Option format

<Name>=<parameters>

Subcommand format

<Name>Δ<parameters>

Examples:

–OUTPUT=lbf	:	option
OUTPUTΔlbf	:	subcommand

In these examples, OUTPUT is the name, and lbf is the parameter.

Notes: 1. A file name consists of three parts: the path name, main file name, and file type. If the file type is omitted, a file type is assumed as follows.

Library file	:	.lib
Object module file	:	.obj
Relocatable load module file	:	.obj
Subcommand file	:	.sub
List file	:	.lst

2. A module name is the name defined in an object module or relocatable load module. In module names, uppercase letters are distinguished from lowercase letters. The pairs of names below, for example, are treated as different names.

Examples: modul1 ←→ MODUL1
abcde ←→ Abcde

Continuation Specification in a Subcommand: When a subcommand is too long to be specified on one line (generally, up to 500 characters per line, but it will depend on the OS), a continuation specifier is used. A continuation specifier is an ampersand (&) at the end of the line. It must always be placed between two parameters; if it is placed within a parameter, it will not be treated as a continuation specifier. Also, if a character (including a space or tab) is typed after the ampersand, an error will occur and the subcommand will not be continued.

In interactive input of subcommands, a hyphen (-) appears as a prompt for further input after continuation has been specified.

Examples:

```
:ADDΔobj00.lib(mod0,mod1),& (RET)
-ibh91,ibh92 (RET)
:ADDΔobj00.lib(mod0,mod1),ob& (RET)
```

Continuation specifier

Specifying continuation in the middle of a parameter generates an error

A subcommand line in a subcommand file can be continued in the same way. The line after a line with the continuation specifier becomes the continuation line.

Example:

Subcommand file

```
DELETEDSUB1,SUB2,& (RET)
sub3 (RET)
```

Continuation specifier

Continuation line

Specifying Comments in a Subcommand File: A comment specifier is used to place notes or other comments in a subcommand file. The specifier is a semicolon (;) placed on a subcommand line, indicating that the rest of the line is a comment. If the semicolon follows a subcommand name or parameter, it must be separated by at least one space or tab.

If the semicolon is placed at the beginning of a subcommand line, the entire line is treated as a comment.

Examples:

```
;EXAMPLE OF LIBRARIAN SUBCOMMAND
```

... the entire line is a comment.

```
LIBRARY∆syslib∆; INDICATES LIBRARY FILE
```

... INDICATES LIBRARY FILE is a comment.

```
ADD∆module.obj;abc
```

... module.obj;abc is treated as a single parameter;
abc is not treated as a comment.

4.2 List of Options and Subcommands

There are 10 options and 15 subcommands, as listed in table 4-1.

Table 4-1 List of Options and Subcommands

No.	Type	Name*1	Function	Opt.*2	Sub.*2	Section
1	File control	LIBRARY	Specifies the library file to be edited	No	Yes	4.3.1
		OUTPUT	Specifies an output library file	Yes	Yes	4.3.2
		DIRECTORY	Specifies directory name replacement	No	Yes	4.3.3
2	Execution control	SUBCOMMAND	Specifies a subcommand file	Yes	No	4.4.1
		CREATE	Creates a library file	Yes	Yes	4.4.2
		ADD	Adds modules	Yes	Yes	4.4.3
		REPLACE	Replaces modules	Yes	Yes	4.4.4
		DELETE	Deletes modules	Yes	Yes	4.4.5
		EXTRACT	Extracts modules	Yes	Yes	4.4.6
		RENAME	Modifies section names	Yes	Yes	4.4.7
		END	End of subcommand input	No	Yes	4.4.8
		EXIT	End of Librarian operations	No	Yes	4.4.9
		ABORT	Aborts Librarian operations	No	Yes	4.4.10
3	List display	LIST	Displays contents of library file	Yes	Yes	4.5.1
		SLIST	Displays section names of library file	Yes	Yes	4.5.2

Notes: 1. The underlined letters of a name are the shortest permissible abbreviated form.
2. The Opt. and Sub. columns indicate whether a name is available as an option or subcommand.

Abbreviating Option and Subcommand Names: Names of options and subcommands can be abbreviated to the point where the name can still be distinguished from other names. As an example, consider the name EXTRACT.

E	: Cannot be distinguished from EXIT or END, so an error occurs.
EX	: Cannot be distinguished from EXIT, so an error occurs.
EXT	: Recognized as EXTRACT.
EXTRA	: Recognized as EXTRACT.
EXTRACT	: Recognized as EXTRACT.
EXTRACTS	: No such name, so an error occurs.

Interrelation among Different Options and Subcommands: Once an option or a subcommand has been specified, other options or subcommands with conflicting functions cannot be specified. This interrelationship is shown in table 4-2.

Table 4-2 Interrelation among Options and Subcommands

	Later Specification of Option/Subcommand														
Specified Option/ Subcommand	**SUBCOMMAND**	**LIBRARY**	**CREATE**	**ADD**	**REPLACE**	**DELETE**	**EXTRACT**	**RENAME**	**OUTPUT**	**DIRECTORY**	**LIST**	**SLIST**	**END**	**EXIT**	**ABORT**
SUBCOMMAND	×	O	O	O	O	O	O	O	O	O	O	O	O	O	O
LIBRARY	O	×	×	O	O	O	O	O	O	O	O	O	O	O	O
CREATE	O	×	×	O	O	O	×	×	×	O	O	O	O	O	O
ADD	O	×	×	O	O	O	×	O	×	O	O	O	O	O	O
REPLACE	O	×	×	O	O	O	×	×	×	O	O	O	O	O	O
DELETE	O	×	×	O	O	O	×	×	×	O	O	O	O	O	O
EXTRACT	O	×	×	×	×	×	O	×	O	O	O	O	O	O	O
RENAME	O	×	×	O	O	O	×	O	×	O	O	O	O	O	O
OUTPUT	O	×	×	×	×	×	O	O	×	O	O	O	O	O	O
DIRECTORY	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
LIST	O	×	×	O	O	O	O	O	O	O	O	O	O	O	O
SLIST	O	×	×	O	O	O	O	O	O	O	O	O	O	O	O
END	O	O	O	×	×	×	×	×	×	O	×	×	×	×	O
EXIT	×	×	×	×	×	×	×	×	×	×	×	×	×	×	O
ABORT	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×

O: Later specification enabled.
× : Later specification disabled, since it conflicts with already specified option or subcommand.

Examples:

```
 lbr (RET)
:LIBRARY∆funclib.lib (RET)
:CREATE∆newlib.lib (RET)
```

A CREATE subcommand cannot be specified after a LIBRARY subcommand. An error occurs, and the CREATE subcommand is ignored.

```
:END (RET)
:LIST (RET)
:EXIT (RET)
```

Specifying a LIST subcommand after an End subcommand generates an error. After END, only the LIBRARY, CREATE, EXIT or ABORT subcommand is valid.

In the following sections, the format below is used to describe each option and subcommand.

				Heading for each option or subcommand
Section number				Section number and heading for option or subcommand
Format	Name	Option	Subcommand	Option or subcommand name and format for specifying parameters
	Parameters			The underlined part of the name is the shortest abbreviated form
Function				Summary of option or subcommand functions
Explanation				Detailed description of functions and restrictions
Examples				Examples of option or subcommand specifications

4.3 File Control

4.3.1 LIBRARY—Specifies the Library File to Be Edited

Format	Name	LIBRARY	Option	Subcommand
			No	Yes
	Parameters	<Library file name>		

Function Specifies an existing library file for editing.

Explanation

(1) This subcommand is specified at the beginning of an editing operation that edits an existing library file or extracts modules from an existing library file.

(2) Only a library file created by this Librarian can be specified.

(3) When no file type is specified as part of the library file name, the type is assumed to be .lib.

(4) This subcommand cannot be used together with the CREATE subcommand, which specifies creation of a new library file.

(5) If, as the result of editing an existing library file, the number of modules becomes zero, the library file will not be updated.

(6) The access right to the updated library file is the same as the access right to a newly created file. Note that the access right prior to the update is not preserved.

Examples

```
LIBRARYΔsyslib
```

Specifies editing of the library file syslib.lib.

4.3.2 OUTPUT—Specifies an Output Library File

Format	Name	OUTPUT	Option	Subcommand
			Yes	Yes
	Parameters	Option	UNIX	<Library file name>
			MS-DOS	<Library file name> {(S) | (U)}
			Subcommand	<Library file name> {(S) | (U)}

Function Specifies a library file for output of extracted modules.

Explanation

(1) Specify the OUTPUT option or subcommand whenever a module is to be extracted from an existing library file.

(2) Specify a new library file name. When no file type is specified as part of the library file name, the type is assumed to be .lib.

(3) The attribute (S) or (U) is assigned to the output file. If unspecified, the attribute is assumed to be (U).

(S) ... System library
(U) ... User library

This attribute determines the order of priority in which library files are searched by the Linkage Editor. A user library has higher search priority. The (S) and (U) parameters cannot be included when OUTPUT is specified as an option in UNIX system.

(4) OUTPUT can be specified either before or after the EXTRACT option or subcommand, which specifies extraction of modules.

(5) OUTPUT cannot be used together with the CREATE, ADD, DELETE, or REPLACE options or subcommands.

(6) When the number of extracted modules is zero, the library file specified by the OUTPUT option or subcommand is not created.

Examples

`-OUTPUT=prog86`

Outputs modules extracted using the EXTRACT subcommand to a file named prog86.lib as a user library.

`OUTPUT∆clib.o(S)`

Outputs modules extracted using the EXTRACT subcommand to a file named clib.o as a system library.

4.3.3 DIRECTORY—Specifies Directory Name Replacement

Format	Name	DIRECTORY	Option	Subcommand
			No	Yes
	Parameters	<Symbol name>(<directory name>)		

Function

Defines a symbol as an alias of a directory. This function enables a long directory name to be input with a simple symbol name.

Explanation

(1) Directory name alias definitionA symbol name is defined as an alias of a directory with the DIRECTORY subcommand.

DIRECTORY Δ <symbol name> (<directory name>)

(2) Directory name referenceTo refer to a directory name, enclose the defined symbol name with a dollar sign ($) and a slash (/) (a dollar sign ($) and a back-slash (\) in MS-DOS system). If the symbol name has not been defined, the Librarian does not replace it with a directory name.

$<symbol name>/ —> Replaced with <directory name>/

(3) Symbol name for up to 16 directory names can be defined.

Examples

```
DIRECTORYΔsymbol(dir1/dir2)
ADDΔ$symbol/file1.obj
```

Defines symbol “symbol” as an alias of directory “dir1/dir2”.Replaces $symbol/dir1/dir2, and as a result, specifies file name dir1/dir2/file1.obj.

4.4 Execution Control

4.4.1 SUBCOMMAND—Specifies a Subcommand File

Format

Name	SUBCOMMAND	Option	Subcommand
		Yes	Yes
Parameters	<Subcommand file name>		

Function

Inputs subcommands from a specified file.

Explanation

(1) Inputs and processes subcommands from a specified subcommand file one at a time.

(2) When no EXIT subcommand is specified, the Librarian waits for command input.

(3) When no file type is specified as part of the file name, the type is assumed to be .sub.

(4) When a SUBCOMMAND option is used together with other options, the SUBCOMMAND is processed last regardless of the option specification order.

Examples

```
-SUBCOMMAND=makelib
```

Inputs subcommands from the subcommand file makelib.sub for use in editing a library file.

4.4.2 CREATE—Creates a Library File

Format	Name	CREATE	Option	Subcommand
			Yes	Yes
	Parameters	Option	UNIX:	<Library file name>
			MS-DOS:	<Library file name> [{ (S) / (U) }]
			Subcommand	<Library file name> [{ (S) / (U) }]

Function Creates a new library file.

Explanation

(1) Specified at the beginning of a group of options or subcommands ending with END or EXIT.

(2) Specify a new library file name. When no file type is specified as part of the library file name, the type is assumed to be .lib.

(3) The attribute (S) or (U) is assigned to the output file. If unspecified, the attribute is assumed to be (U).

(S) ... System library

(U) ... User library

This attribute determines the order of priority in which library files are searched by the Linkage Editor. A user library has higher search priority. The (S) and (U) parameters cannot be included when CREATE is specified as an option in UNIX system.

(4) CREATE cannot be used together with the LIBRARY subcommand.

(5) If the number of modules is zero, no library file is created.

Examples

`-CREATE=userlib.lib`

Creates userlib.lib as a new user library.

`CREATE∆sislib(S)`

Creates sislib.lib as a new system library.

`CREATE∆datax`

Creates datax.lib as a new user library.

4.4.3 ADD—Adds Modules

Format

Name	ADD	Option	Subcommand
		Yes	Yes

Parameters

Option

UNIX:

```
{ <Object module file name>
  <Relocatable load module file name>
  <Library file name> } [{Δ|,}...]
```

MS-DOS:

```
{ <Object module file name>
  <Relocatable load module file name>
  <Library file name>[(<module name>[{Δ|,}...])] } [{Δ|,}...]
```

Sub-command

```
{ <Object module file name>
  <Relocatable load module file name>
  <Library file name>[(<module name>[{Δ|,}...])] } [{Δ|,}...]
```

Function

Adds modules from specified files to a library file.

Explanation

(1) ADD is used to store modules in a new library file, or add modules to an existing library file.

(2) When only a file name is specified, if no file type is specified, the type is assumed to be .obj. When a module name is specified after a file name, the file is assumed to be a library file, so if no file type is specified, the type is assumed to be .lib.

(3) When only certain modules from a library file are to be added, specify the module names after the library file name. Up to 10 module names can be specified. However, module names can not be included when ADD is specified as an option in UNIX system.

Example: ADD lbf (m1,m2,m3)

Module names

Library file name

Explanation (4) When modules in a library file are specified, the specified module names are sorted in alphabetical order and the modules are added in that order. They are not added in the order of specification.

Example: ADD lbf (e, a, d, c, b)
5, 1, 4, 3, 2 ... Order in which modules are added

(5) When the names of modules in a library file are not specified, all modules in the library file are added.

Example: ADD lbf.lib
Library file name

(6) When a module to be added has the same name as a module already in the library file being edited, or when an export symbol defined in the module to be added has the same name as an export symbol in the library file being edited, a warning message is displayed and the module is not added.

(7) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand confirms which modules are stored in a library file.

(8) ADD cannot be used together with the EXTRACT or OUTPUT options or subcommands.

(9) Errors will occur and the parameters after the error occurs will not be processed when:

(a) A specified file does not exist.
(b) A specified module does not exist in a library file.
(c) The content of the specified file is invalid.
(d) The number of modules to be stored exceeds 32,767.
(e) Memory capacity is insufficient to add more modules.
(f) The number of input files exceeds 256.

Examples

`–ADD=mod1,mod2,modx.o`

Adds all modules from the object module files mod1.obj, mod2.obj and modx.o.

`ADD∆iofnc(keyin,crtout)`

Adds the two modules keyin and crtout from the library file iofnc.lib.

`ADD∆syslib.lib`

Adds all modules from the library file syslib.lib.

4.4.4 REPLACE—Replaces Modules

Format

Name	REPLACE	Option	Subcommand
		Yes	Yes

Parameters

Option

UNIX: { <Object module file name> / <Relocatable load module file name> / <Library file name> } [{Δ|,}...]

MS-DOS: { <Object module file name> / <Relocatable load module file name> / <Library file name>[(<module name>[{Δ|,}...])] } [{Δ|,}...]

Sub-command { <Object module file name> / <Relocatable load module file name> / <Library file name>[(<module name>[{Δ|,}...])] } [{Δ|,}...]

Function

Substitutes modules in a specified file for modules of the same name in the library file being edited.

Explanation

(1) When a module in the library file being edited has the same name as a module in the specified file, the former is replaced by the latter. If there is no module with the same name in the library file being edited, the module is simply added.

(2) When only a file name is specified and no file type is specified, the type is assumed to be .obj. When a module name is specified after a file name and no file type is specified, the file is assumed to be a library file and the type is assumed to be .lib.

(3) To substitute only certain modules from a library file, specify the module names after the library file name. Up to 10 module names can be specified. However, module names cannot be included when REPLACE is specified as an option in UNIX system.

Example: REPLACE lbf (m1,m2,m3)

Module names

Library file name

Explanation (4) When modules in library files are specified, the specified module names are sorted in alphabetical order and modules are replaced in that order. They are not replaced in the order of specifications.

Example: REPLACE lbf (e, a, d, c, b)
5, 1, 4, 3, 2 ... Order of replacement

(5) When the names of modules in a library file are not specified, all modules in the file are substituted.

Example: REPLACE lbf.lib
Library file name

(6) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand confirms which modules are stored in a library file.

(7) REPLACE cannot be used together with EXTRACT or OUTPUT options or subcommands.

(8) The following cases will result in error, and the parameters after the error position will not be processed.

(a) A specified file does not exist.
(b) A specified module does not exist in a library file.
(c) The content of the specified file is invalid.
(d) The number of modules to be stored exceeds 32,767.
(e) Memory capacity is insufficient to perform substitution.
(f) The number of input files exceeds 256.

(9) The process of replacing a module involves deleting the module of the same name in the library file being edited, then inputting the module from the file specified by the REPLACE option or subcommand and storing it in the library file. The following special caution is thus required: If a module to be substituted contains an export symbol already defined in another module in the library file, the old module will be deleted, but the replacement module will not be stored.

Examples

`–REPLACE=userlib.lib`

Stores all modules in the library file userlib.lib in the library file being edited, replacing modules with the same name.

`REPLACE∆loadx.rel,loady.rel`

Substitutes the modules in the relocatable load module files loadx.rel and loady.rel for modules of the same name in the library file being edited.

`REPLACE∆datax(member),omf`

Substitutes the module named member in library file datax.lib, and the modules in the object module file omf.obj for modules of the same name in the library file being edited.

4.4.5 DELETE—Deletes Modules

Format	Name	DELETE	Option	Subcommand
			Yes	Yes
	Parameters	<Module name> [{Δ|,}...]		

Function

Deletes specified modules from the library file being edited.

Explanation

(1) If a specified module does not exist in the library file, an error occurs, and the parameters after the error occurrence are not processed.

(2) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand confirms which modules are stored in a library file.

(3) DELETE cannot be used together with EXTRACT or OUTPUT options or subcommands.

Examples

`–DELETE=inchar,outchar`

Deletes the two modules inchar and outchar.

`DELETEΔdatatbl,sort`

Deletes the two modules datatbl and sort.

4.4.6 EXTRACT—Extracts Modules

Format	Name	EXTRACT	Option	Subcommand
			Yes	Yes
	Parameters	<Module name> [{Δ|,}...]		

Function Extracts specified modules from the library file being edited.

Explanation

(1) The extracted modules are output in library file format with the file name specified by the OUTPUT option or subcommand.

(2) The name of an object module or relocatable load module is the name defined in the module. The LIST option or subcommand confirms which modules are stored in a library file.

(3) If a specified module does not exist in the library file, an error occurs, and the parameters after the error occurrence are not processed.

(4) EXTRACT cannot be used together with the CREATE, ADD, DELETE or REPLACE options or subcommands.

Examples

`-EXTRACT=add,sub,mul,div`

Extracts the four modules add, sub, mul, and div from the library file being edited.

`EXTRACTΔalpha,upper,lower,digit,cntrl`

Extracts the five modules alpha, upper, lower, digit, and cntrl from the library file being edited.

4.4.7 RENAME—Modifies Section Names

Format

Name	RENAME	Option	Subcommand
		Yes	Yes
Parameters	<module name>[,...](<section name 1>=<section name 2>[,...])		

Function

Modifies section names in library files in module units.

Explanation

(1) The section names in library files can be modified to freely allocate sections to memory at linkage.

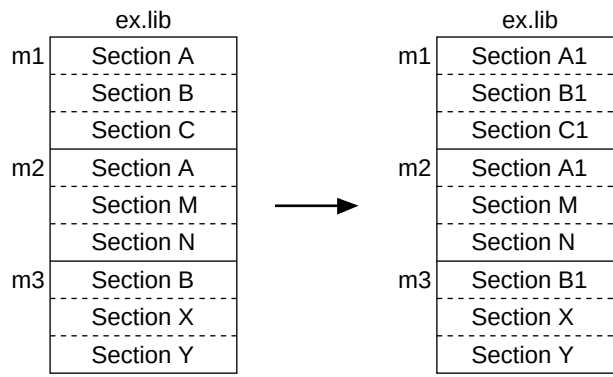
(2) The section names in a library file including a relocatable load module cannot be modified.

(3) When a section name in the module including debugging information is modified, symbols will not be referenced correctly at debugging.

Examples

```
RENAMEΔm1,m2,m3(A=A1,B=B1,C=C1)
```

Modifies sections A, B, and C in module m1 to A1, B1, and C1, section A in module m2 to A1, and section B in module m3 to B1.



4.4.8 END—Specifies End of Subcommand Input

Format	Name	END	Option	Subcommand
			No	Yes
	Parameters	None		

Function Outputs a newly created or updated library file.

Explanation

(1) When more than one library file is edited in one Librarian execution, the editing of each library file is terminated by an END subcommand.

(2) Specification of the END subcommand causes the Librarian to output the edited library file. If, however, the number of modules stored in the library file is zero, the library file is not created or updated.

Examples

```
END
```

Outputs a library file.

4.4.9 EXIT—Specifies End of Librarian Operations

Format

Name	EXIT	Option	Subcommand
		No	Yes
Parameters	None		

Function

Terminates Librarian operations.

Explanation

(1) The EXIT subcommand is used to terminate a set of Librarian operations executed by the subcommand specification.

(2) When executing from a subcommand file, all subcommands following after an EXIT subcommand are ignored. If the EXIT subcommand is not specified, a warning message will be displayed.

(3) When the EXIT subcommand is used, the immediately preceding END subcommand can be omitted. In that case the EXIT subcommand serves also as an END subcommand, causing the library file to be output before terminating the Librarian operation.

Examples

```
EXIT
```

Terminates Librarian operations.

4.4.10 ABORT—Aborts Librarian Operations

Format	Name	ABORT	Option	Subcommand
			No	Yes
	Parameters	None		

Function Aborts Librarian operations.

Explanation

(1) When executing by the subcommand specification, the ABORT subcommand can be used to abort editing operations.

(2) When the ABORT subcommand is specified, the library file being edited will not be created or updated. If, however, a list file was output by a LIST subcommand before the ABORT subcommand, the list file will remain unchanged.

Examples

```
ABORT
```

Aborts Librarian operations.

4.5 List Display

4.5.1 LIST—Displays Contents of a Library File

Format	Name	LIST		Option	Subcommand
				Yes	Yes
	Parameters	Option	UNIX:	[<List file name>]	
			MS-DOS:	[[<List file name>][(S)]]	
		Subcommand	[[<List file name>][(S)]]		

Function Outputs a list of the contents of the library file being edited to the standard output device or to a file.

Explanation

(1) The names of modules stored in the library file, export symbol names, and other information is output on a list. For the list format, see section 6.2, Librarian Lists.

(2) When no list file name is specified, the list is output to the standard output device.

(3) When a list file name is specified, the list is output to a file. Specify a new list file name; the list cannot be appended to an existing file. If an existing file is specified, the existing file contents will be replaced.

(4) When no file type is specified as part of the list file name, the type is assumed to be .lst.

(5) To obtain a list of export symbols designated in modules, specify the (S) parameter. If the (S) parameter is not specified, only the module names will be listed. The (S) parameter cannot be included when LIST is specified as an option in UNIX system.

(6) The LIST option or subcommand can be specified any number of times during the editing process. The library file contents at the point of specification will be listed.

Examples

`-LIST`

Outputs a list to the standard output device.

Export symbols are not shown.

`LIST`

Outputs a list to the standard output device.

Export symbols are not shown.

`LIST∆libx(S)`

Outputs a list including export symbols to a file named libx.lst.

4.5.2 SLIST—Displays Section Names of Library File

Format

Name	SLIST	Option	Subcommand
		Yes	Yes
Parameters	[<List file name>]		

Function

Outputs a list of the contents of the library file being edited to the standard output device or to a file.

Explanation

(1) The names of modules stored in the library file, export symbol names, names of the sections containing export symbol names, and other information is output on a list. For the list format, see section 6.3, Section Name Lists.

(2) When no list file name is specified, the list is output to the standard output device.

(3) When a list file name is specified, the list is output to a file. Specify a new list file name; the list cannot be appended to an existing file. If an existing file is specified, the existing file contents will be replaced.

(4) When no file type is specified as part of the list file name, the type is assumed to be .sct.

(5) The SLIST option or subcommand can be specified any number of times during the editing process. The library file contents at the point of specification will be listed.

Examples

`-SLIST`

Outputs a section name list to the standard output device.

`SLIST∆libx`

Outputs a section name list to a file named libx.sct.

Section 5 Input to the Librarian

5.1 Object Module Files

Object module files output from a C compiler or assembler can be input to the Librarian and stored as modules in library files.

5.2 Relocatable Load Module Files

A relocatable load module file output from the Linkage Editor can be input and stored in a library file as one module.

5.3 Library Files

The Librarian inputs the library file it is editing. Also, modules to be stored in this library file can be input from other library files. Either specified modules can be input, or all the modules in a library file can be input at one time.

Input can be made only from library files created using this Librarian.

Section 6 Output from the Librarian

6.1 Library Files

The Librarian combines two or more modules into a single output library file. It also updates an existing library file, or extracts modules from an existing library file, and outputs the result in library file format.

6.2 Librarian Lists

When the LIST option or subcommand is specified, a list of the library file contents is output to the standard output device or to a file. The format of a librarian list is shown in figure 6-1.

Library file name: (1)

```
Library file name:                    (1)
                                      (1)
Attribute:  (2)
Number of modules:  (3)                    Creation date:  (5)
Number of symbols:  (4)                    Revision date:  (6)
        (7)            (8)                 Entry date:     (9)
    (10)                                        (10)
                :                                :
        (7)            (8)                 Entry date:     (9)
```

Figure 6-1 Librarian List Format

(1) Shows the library file name. If the name is too long to fit on one line it is continued to the next line. When modules are extracted from an existing library file, the list shows the contents of the existing library file.

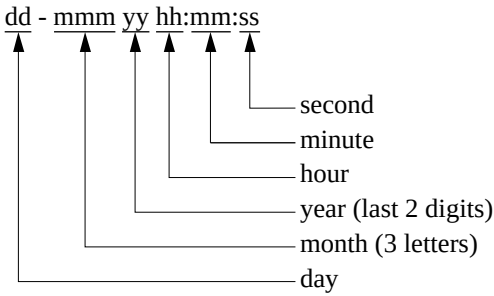
(2) Shows the library file attribute.

SYSTEM: System library
USER: User library

(3) Shows the total number of modules stored in the library file, in decimal notation.

(4) Shows the total number of export symbols in the library file, in decimal notation.

(5) Shows the date and time of library file creation. This information is given in the following format.



(6) Shows the date and time of the most recent library file update. When library files are newly created using the CREATE option or subcommand, this shows the date of creation. The format is the same as for the creation date, above.

(7) Shows the names of modules stored in the library file, in alphabetical order.

(8) Shows the kind of editing operation performed on the module.

BLANK	:	A module stored in an existing library file
(A)	:	An added module
(R)	:	A replacement module
(E)	:	An extracted module

Modules deleted by the DELETE option or subcommand are not listed.

(9) Shows the date and time a module was stored in the library file. The format is the same as for the library file creation date and revision date.

(10) When the (S) parameter is specified with the LIST subcommand, the export symbols in each module are shown. These symbol names are listed in alphabetical order two on each line.

An example of a list when the (S) parameter is specified with the LIST subcommand is given in figure 6-2. Figure 6-3 shows a list without the (S) specification.

```
Library file name:  clib.lib
Attribute:  USER
Number of modules: 6     Creation date:  08-Jan-90 14:18:47
Number of symbols: 6     Revision date:  01-Mar-90 19:56:33

ABS.C                    Entry date:     08-Jan-90 14:18:47
        _abs
ATOF.C                   Entry date:     08-Jan-90 14:18:47
        _atof
ATOI.C                   Entry date:     08-Jan-90 14:18:47
        _atoi
ATOL.C                   Entry date:     08-Jan-90 14:18:47
        _atol
_ALOCBUF             (A) Entry date:     01-Mar-90 19:56:33
        _alcobuf
_DIVI                (A) Entry date:     01-Mar-90 19:56:33
        _divi
```

Figure 6-2 Librarian List (with (S) specification on UNIX)

```
Library file name:  clib.lib
Attribute:  USER
Number of modules: 6     Creation date:  08-Jan-90 14:18:47
Number of symbols: 6     Revision date:  01-Mar-90 19:56:33

ABS.C                    Entry date:     08-Jan-90 14:18:47
ATOF.C                   Entry date:     08-Jan-90 14:18:47
ATOI.C                   Entry date:     08-Jan-90 14:18:47
ATOL.C                   Entry date:     08-Jan-90 14:18:47
_ALOCBUF             (A) Entry date:     01-Mar-90 19:56:33
_DIVI                (A) Entry date:     01-Mar-90 19:56:33
```

Figure 6-3 Librarian List (no (S) specification on UNIX)

6.3 Section Name Lists

When the SLIST option or subcommand is specified, a list of the section contents of the library file are output to the standard output device or to a file. The format of a section name list is shown in figure 6-4.

```
Library file name:                    (1)
                                      (1)
Attribute:  (2)
Number of modules:  (3)               Creation date:  (5)
Number of symbols:  (4)               Revision date:  (6)
        (7)                           Entry date:  (8)
            (9)                           (10)
              :                             :
        (7)                           Entry date:  (8)
```

Figure 6-4 Section Name List Format

(1) Shows the library file name. If the name is too long to fit on one line it is continued to the next line. When modules are extracted from an existing library file, the list shows the contents of the existing library file.

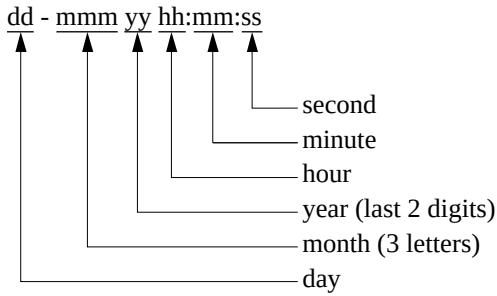
(2) Shows the library file attribute.

SYSTEM: System library
USER: User library

(3) Shows the total number of modules stored in the library file, in decimal notation.

(4) Shows the total number of export symbols in the library file, in decimal notation.

(5) Shows the date and time of library file creation. This information is given in the following format.



(6) Shows the date and time of the most recent library file update. When library files are newly created using the CREATE option or subcommand, this shows the date of creation. The format is the same as for the creation date, above.

(7) Shows the names of modules stored in the library file, in alphabetical order.

(8) Shows the date and time a module was stored in the library file. The format is the same as for the library file creation date and revision date.

(9) Shows the export symbols in each module.

(10) Shows the name of the section containing the export symbol name.

An example of a list specified with the SLIST subcommand is given in figure 6-5.

```
Library file name:  clib.lib
Attribute:  USER
Number of modules: 6     Creation date:  8-Jan-90 14:18:47
Number of symbols: 6     Revision date:  01-Mar-90 19:56:33

ABS.C                    Entry date:     08-Jan-90 14:18:47
        _abs                    P
ATOF.C                   Entry date:     08-Jan-90 14:18:47
        _atof                   P
ATOI.C                   Entry date:     08-Jan-90 14:18:47
        _atoi                   P1
ATOL.C                   Entry date:     08-Jan-90 14:18:47
        _atol            CODE
_ALOCBUF                 Entry date:     01-Mar-90 19:56:33
        _alcobuf                P
_DIVI                    Entry date:     01-Mar-90 19:56:33
        _divi                   P2
```

Figure 6-5 Section Name List

6.4 Console Messages

The Librarian displays the following messages on the standard output device.

Opening Message: Displayed when the librarian command is input.

```
H SERIES  OBJECT  LIBRARIAN  Ver. 1.4
Copyright (C) Hitachi, Ltd. 1988
Licensed Material of Hitachi, Ltd.
```

Normal Completion Message: Displayed when library file editing has ended normally.

```
OBJECT  LIBRARIAN  COMPLETED
```

Abort Message: Displayed when the library file editing is aborted by either an error or an ABORT subcommand.

```
OBJECT  LIBRARIAN  ABORT
```

Subcommand Prompt: Indicates that the Librarian is in subcommand input wait state during interactive execution.

```
:
```

Subcommand Continuation Symbol: Request for a continuation line, when continuation of a subcommand is specified during interactive execution.

```
-
```

Section 7 Error Messages

The Librarian outputs error messages in the following form.

** <Error number> <Error message> [(<Additional information>)]

Error Number: The first digit indicates the level of the error. (xx represents the second and third digits.)

1xx : Warning : Processing of a particular module is skipped.

2xx : Error : If started by input from the command line or a subcommand file, processing is stopped. In interactive mode, processing of the subcommand is stopped when the error is detected, and a prompt is displayed for the next subcommand.

3xx : Fatal error : Processing is stopped.

A list of error messages is given below in tables 7-1, 7-2 and 7-3, in the following format.

Error number	Error message	Additional information
	Description of error	
	Corrective action, etc.	

Note: Additional information includes the name of the file in which the error occurred, or the module name or symbol name. In the list of errors, — means that no additional information is given.

Table 7-1 List of Warning Messages

101	DUPLICATE MODULE	Module name
	An attempt was made to add a module already stored in the library file.	
	Processing of the module is skipped.	
102	DUPLICATE SYMBOL	Module name ** Symbol name
	An attempt was made to add an export symbol already present in the library file.	
	Processing of the module is skipped.	
103	IDENTIFIER CHARACTER EXCEEDS 251	Module name
	A module name of more than 251 characters was specified.	
	The name is valid up to the 251th character. The rest is ignored.	
104	EXIT SUBCOMMAND NOT FOUN—ASSUMED	—
	No EXIT subcommand was specified.	
	Processing continues as though an EXIT subcommand had been specified.	
105	SUBCOMMAND LINE LENGTH TOO LONG	—
	Symbols are replaced with the corresponding directory names, and the file name exceeds 511.	
	The file name is valid up to the 511th character.	
106	TOO MANY DIRECTORY COMMANDS	—
	More than 16 directory names have been specified with the DIRECTORY subcommand.	
	Up to 16th specification is valid.	
107	MODULE COUNT 0	—
	The total number of modules becomes zero.	
	Processing is terminated. Check the specification for editing modules.	
108	SECTION NOT FOUND	Module name ** Section name
	The specified section cannot be found.	
	Check the section name and respecify it.	
109	CANNOT PRINT SECTION LIST	Module name
	The SLIST option or subcommand is specified for the file containing a relocatable load module.	
	Specify the SLIST option or subcommand only for absolute modules.	
110	CANNOT RENAME SECTION NAME	Module name
	The RENAME option or subcommand is specified for the file containing a relocatable load module.	
	Specify the RENAME option or subcommand only for absolute modules.	

Table 7-2 List of Error Messages

201	INVALID SUBCOMMAND/OPTION	—
	The option or subcommand specified is invalid in this context.	
	Specify a valid option or subcommand.	
202	SYNTAX ERROR	—
	Syntax of the specified option or subcommand is incorrect.	
	Check the syntax and respecify the option or subcommand.	
203	SUBCOMMAND LINE LENGTH TOO LONG	—
	Length of the subcommand entry exceeds 128 characters.	
	Respecify, keeping the length within 128 characters.	
204	CONFLICTING SUBCOMMAND	—
	Subcommands are specified in the wrong order, or an illegal combination of subcommands is specified.	
	Check the order of subcommands and respecify.	
205	ILLEGAL FILE NAME	—
	The specified file name is not valid.	
	Specify a correct file name.	
206	ILLEGAL MODULE NAME	—
	The specified module name is not valid.	
	Specify a correct module name.	
207	MODULE NOT FOUND	Module name
	The specified module cannot be found.	
	Check the name of the module, then respecify.	
208	MISSING OUTPUT FILE NAME	—
	No output file was specified with the EXTRACT option or subcommand.	
	Use the OUTPUT option or subcommand to specify an output file.	
209	TOO MANY INPUT FILES	—
	More than 12 input files were specified for input at the same time.	
	First output the library file, then re-input the library file and input the remaining files.	
210	TOO MANY MODULES	—
	The number of modules exceeds the allowable number.	
	No more modules can be stored in the library file now being created or edited. Store any additional modules in a separate library file.	

Table 7-2 List of Error Messages (cont)

211	TOO MANY SYMBOLS	—
	The number of symbols exceeds the allowable number.	
	The library file now being created or edited cannot contain any more symbols. Modules with additional symbols must be stored in a separate library file.	
212	ILLEGAL FILE FORMAT	—
	The specified file format is incorrect.	
	Check the file contents and re-execute.	
213	MEMORY OVERFLOW	—
	There is no space remaining in the Librarian's usable memory.	
	Obtain additional memory and re-execute.	
214	FILE NOT FOUND	File name
	The specified file cannot be found.	
	Check the directory and the specified file name, then respecify.	
215	DUPLICATE SECTION	Module name ** Section name
	The specified section is in a module.	
	Check the section name and respecify it.	
216	ILLEGAL SECTION NAME	—
	The specified section name is illegal.	
	Check the section name and respecify it.	

Table 7-3 List of Fatal Error Messages

301	INVALID COMMAND PARAMETER	—
	An improper command parameter was specified.	
	Check the command parameters and re-execute.	
302	CONFLICTING OPTION	—
	There is a contradiction among different options specified.	
	Check the order of option specification, then respecify.	
303	CANNOT OPEN FILE	File name
	File cannot be opened, or the CREATE or OUTPUT option or subcommand specified an already existing file.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Correct the problem, then re-execute. If an existing file was specified by the CREATE or OUTPUT option or subcommand, delete the existing file, then re-execute.	
304	CANNOT INPUT FILE	File name
	File cannot be input.	
	Check the specified file name. If the file name is correct, there may be a disk hardware error. Correct the problem, then re-execute.	
305	CANNOT OUTPUT FILE	File name
	File cannot be output.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Correct the problem, then re-execute.	
306	CANNOT CLOSE FILE	File name
	File cannot be closed.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware error. Correct the problem, then re-execute.	
307	CANNOT READ	—
	Because forcible termination was specified, processing is aborted.	
	Re-execute the processing.	
308	MEMORY OVERFLOW	—
	The memory space is insufficient for the librarian.	
	Check the operating environment and re-execute the processing.	

Note: In the UNIX system, the Librarian uses temporary files with names in the format shown below. These temporary file names may appear as additional information in error messages.

Annnnn.TEMP

5 digits, decimal

Section 8 Restrictions

Restriction on the Librarian are shown in table 8-1. If the numerical restrictions are exceeded, Librarian operations will not operate correctly.

Table 8-1 Restrictions on Librarian Processing

No.	Item	Limits	Remarks
1	The number of modules that can be stored in a library file	32,767 max.	Assumes that the system on which Librarian runs has adequate memory.
2	The number of symbols that can be present in a library file	65,535 max.	
3	The number of input files	256 max.	Total number of files specified by LIBRARY, ADD, or REPLACE not including subcommand files.
4	The number of modules that can be specified in a library file	10 max.	When specifying a library file with ADD or REPLACE.
5	Length of file name	128 characters max.	Includes default file-type characters. File name format depends on OS.
6	Length of module name	251 characters max.	
7	Length of symbol name	251 characters max.	
8	Input file formats	• Object module file output by assembler or C compiler. • Relocatable load module file. • Library file created using this Librarian.	

Appendix A Examples of Librarian Usage

A.1 Librarian Execution by Command Line

```
lbr∆-CREATE=func-ADD=abs,mod,sqrt,exp,log (RET)
```
...(1) Creation

(a) (b)

```
lbr∆func-ADD=sin,cos-DELETE=abs,mod-LIST (RET)
```
...(2) Editing

(c) (d) (e) (f)

```
lbr∆func-EXTRACT=sqrt,exp-OUTPUT=newfnc (RET)
```
...(3) Extraction

(g) (h) (i)

(a) The CREATE option is specified at the beginning of the option line to create a new library file.

(b) The file names for the modules to be entered are specified using the ADD option.

(c) The name of the library file to be edited is specified.

(d) The file names for modules to be added to the existing library file are specified using the ADD option.

(e) The names of the modules to be deleted from the existing library file are specified using the DELETE option.

(f) The LIST option is specified to confirm the editing results.

(g) An existing library file from which modules are to be extracted is specified.

(h) The names of the modules to be extracted are specified using the EXTRACT option.

(i) The name of a new library file to which the extracted modules are to be output is specified using the OUTPUT option.

This process is illustrated in figure A-1.

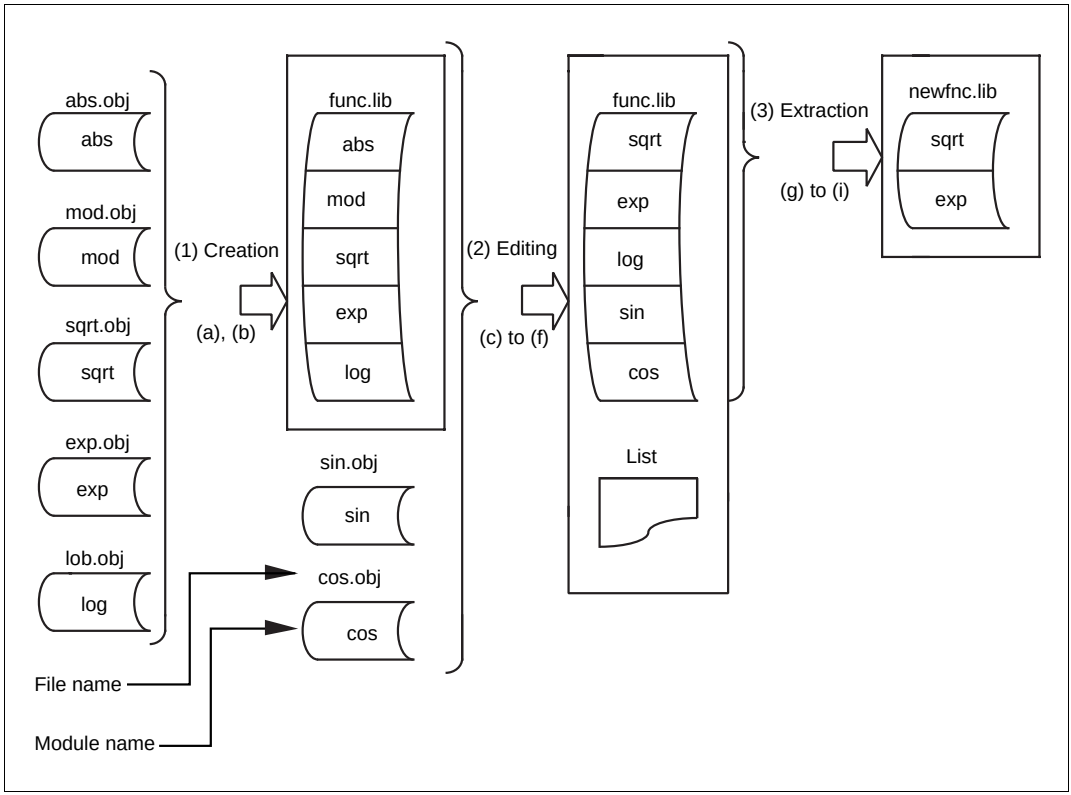


Figure A-1 Results of Librarian Execution by Command Line

A.2 Librarian Execution by Subcommands

```
lbr (RET)                                  ...(a)
CREATE∆func (RET)                          ...(b)  ⎫
ADD∆sqrt,exp,log,sin,cos (RET)             ...(c)  ⎬ (1) Creation
END (RET)                                  ...(d)  ⎭
LIBRARY∆func (RET)                         ...(e)  ⎫
REPLACE∆sin.new,cos.new,tan.new (RET)      ...(f)  ⎬ (2) Editing
END (RET)                                  ...(g)  ⎭
LIBRARY∆func (RET)                         ...(h)  ⎫
LIST (RET)                                 ...(i)  ⎪
EXTRACT∆sqrt,exp (RET)                     ...(j)  ⎬ (3) Extraction
OUTPUT∆newfnc (RET)                        ...(k)  ⎪
END (RET)                                  ...(l)  ⎭
EXIT (RET)                                 ...(m)
```

(a) The Librarian is started.

(b) The CREATE subcommand is specified at the beginning of the option line to create a new library file.

(c) The file names of modules to be loaded are specified using the ADD subcommand.

(d) The END subcommand is specified to terminate the creation process.

(e) The name of the library file to be edited is specified.

(f) Modules in the existing library file are replaced, using the REPLACE subcommand. The file names of the modules to be replaced is specified.

(g) The END subcommand is specified to terminate the editing process.

(h) An existing library file is designated for extraction of modules.

(i) The LIST subcommand is specified to confirm the contents of the existing library file.

(j) The names of the modules to be extracted are specified using the EXTRACT subcommand.

(k) The name of a new library file to which the extracted modules are to be output is specified using the OUTPUT subcommand.

(l) The END subcommand is specified to terminate the extraction process.

(m) The EXIT subcommand is specified to terminate the Librarian program.

This process is illustrated in figure A-2.

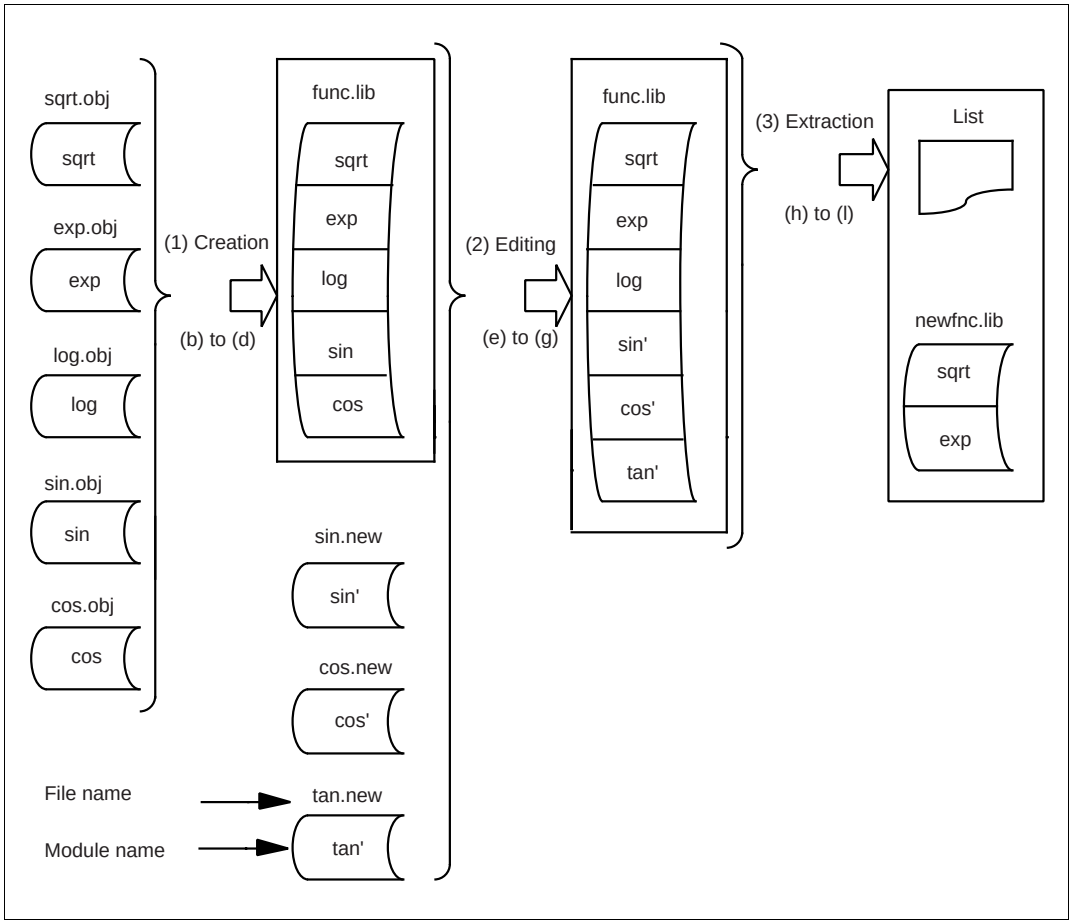


Figure A-2 Results of Librarian Execution by Subcommand

Appendix B Note on Librarian Usage in MS-DOS System

Before using this Librarian, set the MS-DOS configuration file (CONFIG.SYS) with the editor as follows.

```
FILES=20                    (1)
SHELL=a:\command.com a:\    (2)
/p
```

(1) The number of files that is allowed to open at one time during Librarian operation.

(2) Directory path specification that is required when COMMAND.COM is reloaded.

Part III

Object Converter Guide

Section 1 Object Format Conversion

To input the load modules output by the Linkage Editor into an emulator or PROM programmer, they must first be converted to S-type object format using the Object Format Converter.

1.1 Executing the Object Format Conversion

The command line format for starting the Object Format Converter is as follows.

```
cnvs∆<Input file name>[∆<output file name>] (RET)
```

For details on file names, refer to appendix B, File Name Specifications, in Part I, Linkage Editor Guide.

Command Name: The Object Format Converter is started up by specifying the command “cnvs.”

Input File Name: The name of an absolute-format load module file to be input to the Object Format Converter is specified. Relocatable load module files cannot be specified.

If the file type is omitted from the file name, the Object Format Converter automatically assumes this to be “.abs” when it inputs the file.

Output File Name: The name of the S-type object file to be output by the Object Format Converter is specified. If the file type is omitted from the file name, the Object Format Converter automatically assumes this to be “.mot” when it outputs the file.

Examples of command line specification are given below.

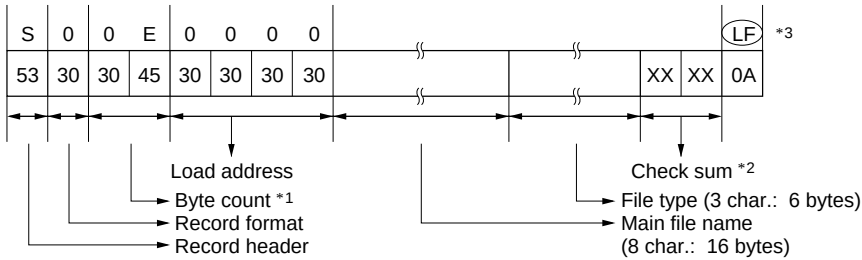
```
cnvs∆prog1.lmd∆prog1.sty (RET) ....     (1)
cnvs∆prog1∆prog1 (RET) ............     (2)
```

(1) File “prog1.lmd” is input, and file “prog1.sty” is output.

(2) File “prog1.abs” is input, and file “prog1.mot” is output.

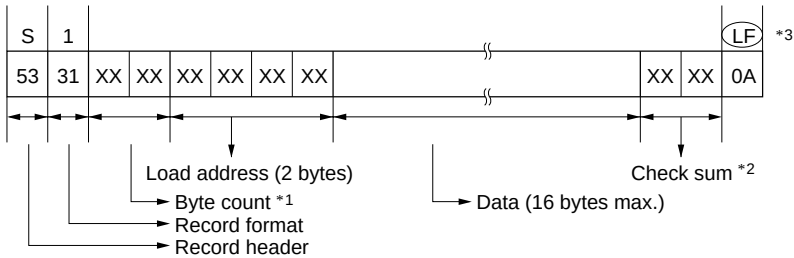
The S-type object format is shown in figure 1-1.

(a) Header record (S0 record)

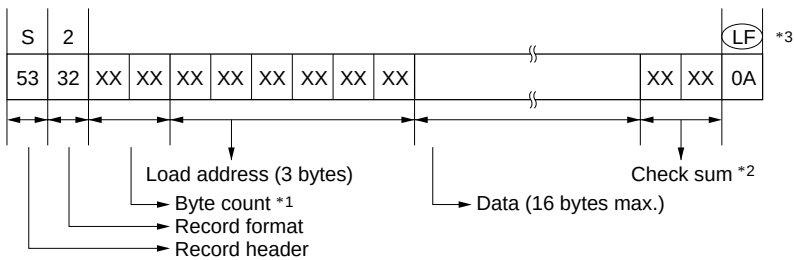


(b) Data record (S1, S2, and S3 record)

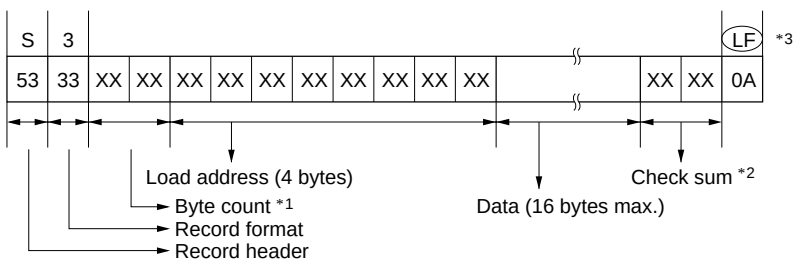
(i) When load address is between 0 and 0FFFF (hexadecimal)



(ii) When load address is between 10000 and 0FFFFFF (hexadecimal)



(iii) When load address is between 1000000 and 0FFFFFFFF (hexadecimal)

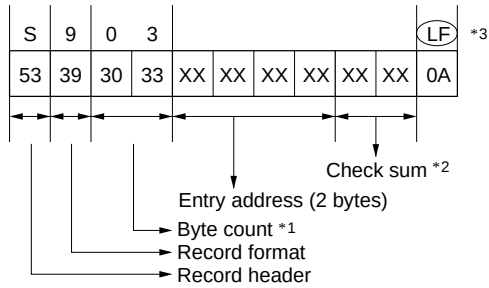


Notes: 1. The byte count is the number of bytes from the load address (or entry address) to check sum.
2. Check sum is the 1's complement of the result of adding the data values from the byte count to that before check sum, in byte units.
3. **"LF"** indicates the line feed code.

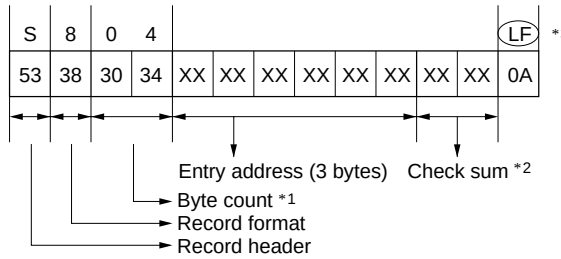
Figure 1-1 S-Type Object Format

(c) End record (S9, S8, and S7 record)

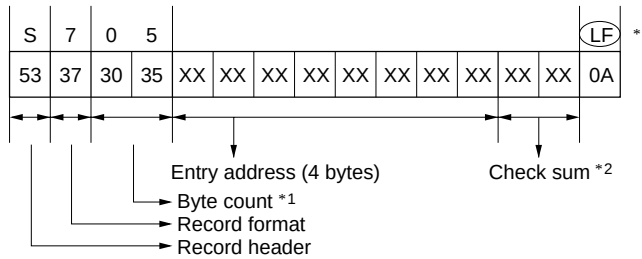
(i) When load address is between 0 and 0FFFF (hexadecimal)



(ii) When load address is between 10000 and 0FFFFFF (hexadecimal)



(iii) When load address is between 1000000 and 0FFFFFFFF (hexadecimal)



Notes:
1. The byte count is the number of bytes from the load address (or entry address) to the check sum.
2. The check sum is the 1's complement of the sum of the data values from the byte count to the byte before the check sum, in byte units.
3. "LF" indicates the line feed code.

Figure 1-1 S-Type Object Format (cont)

1.2 Error Messages

When errors are made in command specification, or when an error is detected during the conversion process, the Object Format Converter outputs error messages in the following format.

```
**Δ <Error number> Δ <error message>[(<additional information>)]
```

1st column

A list of error messages is given below in table 1-1 in the following format.

Error Number	Error Message	Additional Information
	Nature of Error	
	Converter actions and corrective actions	

Notation used in table: —: No additional information

Table 1-1 Object Format Converter Error Messages

301	INVALID COMMAND PARAMETER	—
	An improper command parameter was specified.	
	Check the command parameters and re-execute.	
302	FILE NOT FOUND	File name
	The specified file cannot be found.	
	Check the directory and the specified file name, then re-execute.	
303	CANNOT OPEN FILE	File name
	File cannot be opened.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
304	CANNOT READ FILE	File name
	File cannot be input.	
	Check the specified file name. If the file name is correct, there may be a disk hardware problem. After checking the problem, re-execute.	
305	CANNOT WRITE FILE	File name
	File cannot be output.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
306	CANNOT CLOSE FILE	File name
	File cannot be closed.	
	Check the specified file name. If the file name is correct, the disk may be full, or there may be a disk hardware problem. After checking the problem, re-execute.	
307	ILLEGAL FILE FORMAT	File name
	The specified file format is incorrect.	
	Check the file contents, then re-execute.	
308	ILLEGAL FILE NAME	File name
	An illegal file name was specified.	
	Specify a correct file name.	
309	MEMORY OVERFLOW	—
	Insufficient memory is available for use by the Object Format Converter.	
	Expand the memory or revise the user program, then re-execute.	

Index

W

H Series Linkage Editor, Librarian, and Object Converter User's Manual

Publication Date: 1st Edition, October 1996
Published by: Semiconductor and IC Div.
Hitachi, Ltd.
Edited by: Technical Documentation Center
Hitachi Microcomputer System Ltd.