User's Manual

CC-RL

Compiler

User's Manual

Applicable Revision

V1.00.00 to V1.13.00

Target Device

RL78 Family

Target CPU Cores:

RL78-S1, RL78-S2, RL78-S3

Renesas Electronics
www.renesas.com

Rev.1.13 2023.12

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

How to Use This Manual

This manual describes the role of the CC-RL compiler for developing applications and systems for RL78 family, and provides an outline of its features.

Readers	This manual is intended for users who wish to understand the functions of the CC-RL and design software and hardware application systems.
Purpose	This manual is intended to give users an understanding of the functions of the CC-RL to use for reference in developing the hardware or software of systems using these devices.
Organization	This manual can be broadly divided into the following units. 1. GENERAL 2. COMMAND REFERENCE 3. OUTPUT FILES 4. COMPILER LANGUAGE SPECIFICATIONS 5. ASSEMBLY LANGUAGE SPECIFICATIONS 6. SECTION SPECIFICATIONS 7. LIBRARY FUNCTION SPECIFICATIONS 8. STARTUP 9. FUNCTION CALL INTERFACE SPECIFICATIONS 10. MESSAGE 11. CAUTIONS A. QUICK GUIDE
How to Read This Manual	It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	XXX (overscore over pin or signal name)
Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remarks:	Supplementary information
Numeric representation:	Decimal ... XXXX Hexadecimal ... 0xXXXX

TABLE OF CONTENTS

1. GENERAL

This document is the user's manual for the RL78 family's C compiler CC-RL V1.00 to V1.13.
This chapter provides a general outline of CC-RL.

1.1 Outline

CC-RL is a program that converts programs described in C language or assembly language into machine language.

1.2 Special Features

CC-RL is equipped with the following special features.

(1) Language specifications in accordance with C90 and C99 standards.
The C language specifications conform with the C90 and C99 standards.

(2) Advanced optimization
Advanced optimization including global program optimization is applied.
This yields smaller and faster code.

(3) High portability
The industry-standard ELF format is used for object code. In addition, the industry-standard DWARF2 or DWARF3 format is used for debugging information.

(4) Multifunctional
Static analysis and other functionality is provided via linking between Renesas Electronics and partner tools.

1.3 Copyrights

This software uses the following software products:
- LLVM and CLANG are copyrighted by University of Illinois at Urbana-Champaign.
- Protocol Buffers is copyrighted by Google Inc.

The library for C++ uses the following software products. For details, see the license files included in the compiler.
- compiler_rt
- libc++
- libc++abi
- newlib

Other software components are copyrighted by Renesas Electronics Corporation.

1.4 License

A license manager manages licenses to the compilers.
If you have a license, the compiler will operate as the Standard or Professional edition depending on the license you are using.
Refer to section 1.5 Standard and Professional Editions, for more on the Standard and Professional editions.
If the license manager is not able to recognize a Standard or Professional license, the compiler operates as the free evaluation edition.
Refer to section 1.6 Free Evaluation Editions, for more on the free evaluation edition.
For details of the licenses and the license manager, refer to the User's Manual of the License Manager.
Use V2.00 or later versions of the license manager for V1.04 and later versions of CC-RL.

1.5 Standard and Professional Editions

There are two editions of the compilers, the Standard and the Professional editions.
The Standard editions support C90- and C99-compliant C-language specifications, and also provide the essential features for writing programs for embedded systems.
As well as the features of the Standard editions, the Professional editions have additional features which help to improve the quality of the customer's programs and shorten development periods.
The additional features of Professional editions are available through compiler options, #pragma directives and libraries.
For descriptions of the options only available for the Professional editions, refer to Table 2.2 Compile Options, or the descriptions of the individual options.

For descriptions of the #pragma directives that only the Professional editions support, refer to Table 4.15 List of Supported #pragma Directive.

See "7.1 Supplied Libraries" for libraries that are only supported by the Professional Edition.

1.6 Free Evaluation Editions

The free evaluation editions have a trial period of 60 days from the day of the first building by the compiler over which you can use features equivalent to those of the Professional editions.

After that period, the additional features of the Professional editions are no longer available, and the following restrictions apply.

- [V1.11 or earlier] The section sizes that can be allocated to the ROM area is up to 64 Kbytes in total. A linker error occurs when the size exceeds 64 Kbytes.
- [V1.12 or later] The available optimization levels are -Onothing and -Olite only. There is no restriction on the section size.

The version number of the optimizing linkage editor is prefixed by W while a compiler is operating as an evaluation edition and by V when it is operating as a paid edition.

Examples are given below.

- Version of a free evaluation edition:
  Renesas Optimizing Linker W1.01.01 [25 Apr 2014]
- Version of a paid edition:
  Renesas Optimizing Linker V1.01.01 [25 Apr 2014]

We do not supply the following services for the evaluation editions.
Consider purchasing a paid edition if you require them.

- Technical support
- E-mail delivery of items such as information on revisions

2. COMMAND REFERENCE

This section describes the detailed specifications of each command included in CC-RL.

2.1 Overview

CC-RL generates files executable on the target system from source programs described in C language or assembly language.

CC-RL consists of the following commands. A single driver (ccrl) controls all phases from compilation to linking. It also creates standard libraries by using the library generator (lbgrl).

ccrl: Compilation driver start command
asrl: Assembler start command
rlink: Optimizing linker start command
lbgrl: Library generator [V1.13 or later]

Processing of each command is shown below.

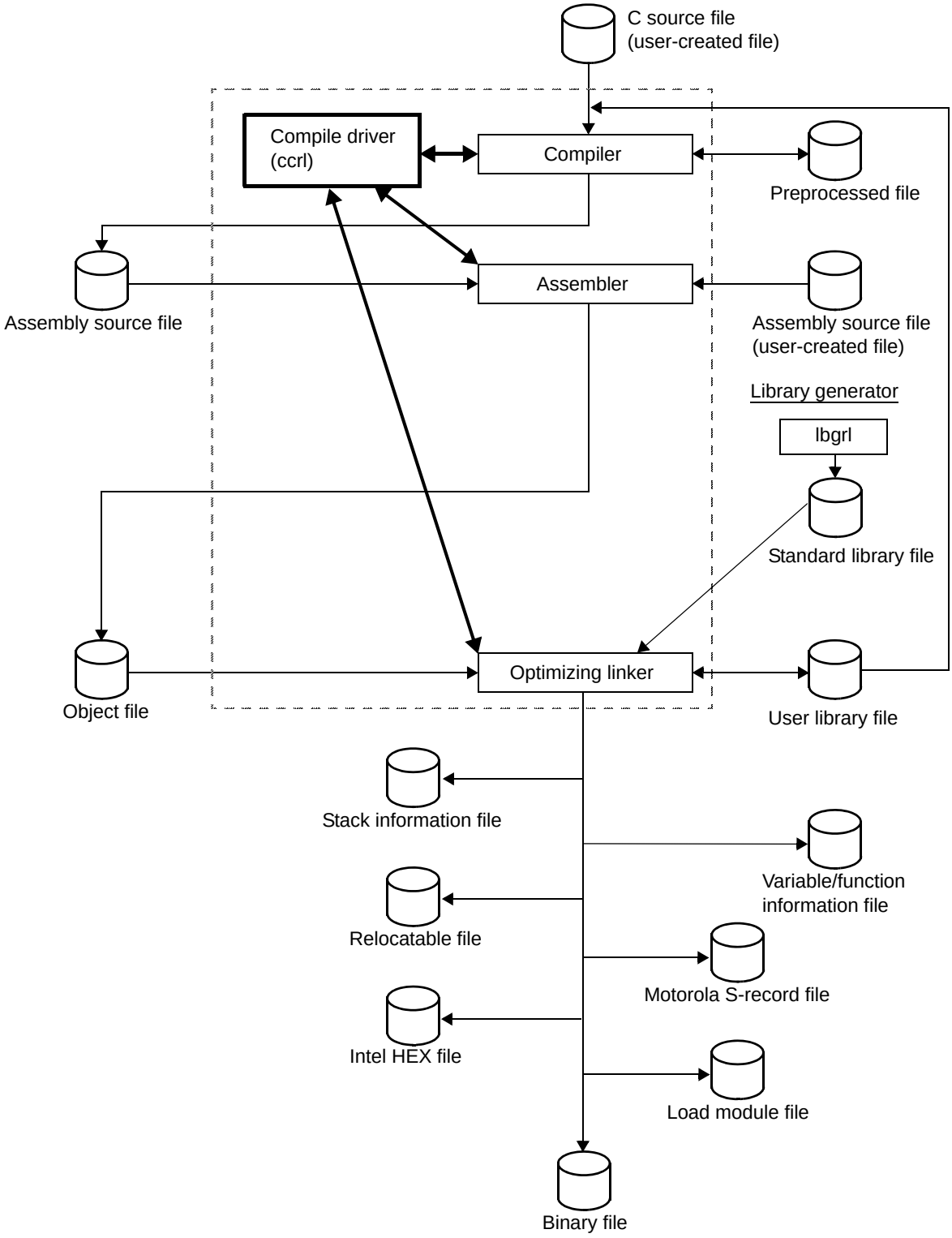
(1) Compiler
Performs processing of preprocess directives, comment processing, and optimization for a C source program and then generates an assembly source file.

(2) Assembler
Converts an assembly source program into machine language instructions and then generates a relocatable object file.

(3) Optimizing linker
Links object files and library files to generate object files (load module files) that are executable on the target system.
It also handles the creation of ROM images for use in embedded applications, optimization during the linking of relocatable files, the creation and editing of library files, conversion to Intel HEX files and Motorola S-record files, and the generation of variable/function information files containing declarations of **saddr** variables and **callt** functions.

(4) Library generator [V1.13 or later]
A tool that generates standard libraries. When creating a standard library, any compile options can be specified.

Figure 2.1 Operation Flow of ccrl



2.2 I/O Files

The I/O files of the ccrl command are shown below.

Table 2.1 I/O Files of ccrl Command

File Type	Extension	I/O	Description
C source file	.c	I	Source file described in C language This is created by the user.
C++ source file	.cpp, .cc, .cp	I	Source file described in C++ language This is described by the user.
Preprocessed file	.i[Note 1]	O	Text file which the execution result of preprocess processing for the input file is output This file is output when the -P option is specified.
Assembly source file	.asm[Note 1]	O	Assembly language file generated from C source file by compilation This file is output when the -S option is specified.
	.asm .s	I	Source file described in assembly language This is created by the user.
Header file	free	I	File referred by source files This file is described in C language or assembly language. This is created by the user. The extension is free, but the following is recommended. - #include directive: .h - $include control instruction: .inc
Object file	.obj[Note 1]	I/O	ELF-format file including machine-language information, relocation information relating to machine-language allocation addresses, and symbol information
Assemble list file[Note 2]	.prn[Note 1]	O	List file which has information from the assemble result This file is output when the -asmopt=-prn_path option is specified.
Library file	.lib[Note 1]	I/O	ELF-format file in which two or more object files are included This file is output when the -lnkopt=-form=library option is specified.
Library backup file	.lbk	O	This file saves the contents of existing library files before they are overwritten by the library generator.
Load module file	.abs[Note 1]	I/O	ELF-format file of the object code of the link result This is the input file when a hex file is output. This file is output when the -lnkopt=-form=absolute option is specified. If you specify the -lnkopt option but not the -form option, the command assumes that the above option has been specified.
Relocatable file	.rel[Note 1]	I/O	Relocatable object file This file is output when the -lnkopt=-form=relocate option is specified.
Intel HEX file[Note 2]	.hex[Note 1]	I/O	Load module file converted into the Intel HEX format This file is output when the -lnkopt=-form=hexadecimal option is specified.
Motorola S-record file[Note 2]	.mot[Note 1]	I/O	Load module file converted into the Motorola S-record This file is output when the -lnkopt=-form=stype option is specified.

File Type	Extension	I/O	Description
Binary file	.bin[Note 1]	I/O	Load module file converted into the binary format This file is output when the -lnkopt=-form=binary option is specified.
Symbol address file	.fsy	I/O	Assembly source file where external defined symbols are described in assembler directives This file is output when the -lnkopt=-fsymbol option is specified.
Link map file[Note 2]	.map[Note 1]	O	List file which has information from the link result This file is output when the -lnkopt=-list option is specified.
Library list file[Note 2]	.lbp[Note 1]	O	List file which has information from the library creation result This file is output when the -lnkopt=-list option is specified.
Stack information file	.sni	O	List file which has information of the stack capacity This file is output when the -lnkopt=-stack option is specified.
Variable/function information file	.h[Note 1]	I/O	File containing declarations of the saddr variable or callt function This file is output when the -lnkopt=-vfinfo option is specified.
Static analysis information file	free	I/O	File which has information from the static analysis result The extension is free, but ".cref" is recommended. This file is output when the -cref option is specified.
Error message file	free	O	File which contains error messages The extension is free, but ".err" is recommended. This file is output when the -error_file option is specified.
Subcommand file	free	I	File which contains the parameters of the execution program This is created by the user.
Tool usage information file	.ud .udm	O	File which is output for collecting tool usage information

Note 1. The extension can be changed by specifying the option.

Note 2. See "3. OUTPUT FILES" for details about each file.

2.3 Environment Variable

This section explains the environment variables.
The environment variables of the optimizing linker and the examples when specifying them on the command line are shown below.

- HLNK_LIBRARY1, HLNK_LIBRARY2, HLNK_LIBRARY3
  Specify the default library that the optimizing linker uses.
  The library specified by the -library option has the precedence for linking.
  After that, if unresolved symbols remain, default libraries HLNK_LIBRARY1, HLNK_LIBRARY2, and HLNK_LIBRARY3 are searched in that order.

  Example

```
>set HLNK_LIBRARY1=usr1.lib
>set HLNK_LIBRARY2=usr2.lib
>set HLNK_LIBRARY3=usr3.lib
```

- HLNK_TMP
  Specify the folder where the optimizing linker creates temporary files.
  If this environment variable is not specified, the files are created in the current folder.

  Example

```
>set HLNK_TMP=D:\workspace\tmp
```

- HLNK_DIR
  Specify the folder where the input files for the optimizing linker are stored.
  The files specified by the -input and -library options are searched from the current folder and the folder specified by HLNK_DIR in that order.
  However, the files specified with wildcard characters are searched in the current folder.

  Example

```
>set HLNK_DIR=D:\workspace\obj1;D:\workspace\obj2
```

2.4 Method for Manipulating

This section explains how to manipulate each command.

- Command line operation
- Subcommand file usage

2.4.1 Command line operation

The compilation driver (ccrl) identifies the extension of the input file, and then starts the compiler, assembler, and linker. Alphabetical extensions are not case sensitive.

- If the input file has the extension .s or .asm, the compilation driver assumes it is an assembly language file, and starts the assembler.
- If the input file has the extension .c, the compilation driver assumes it is a C source file, and starts the compiler.
- If the input file has the extension .cpp, .cc, or .cp, the compilation driver assumes that it is a C++ source file, and starts the compiler.
- If the input file has the extension .obj, the compilation driver assumes that it is an object file, and starts the linker.

If the input file has an extension other than the above, the compilation driver assumes that the file is a C source file, and starts the compiler.
You can specify the language standard of C source files and C++ source files by using the -lang option.

(1) Specification format
Enter the following on the command line.

```
>ccrl[Δoption]...Δfile[Δfile|Δoption]...
```

```
>asrl[Δoption]...Δfile[Δfile|Δoption]...
```

```
>rlink[{Δfile|Δoption}...]
```

```
>lbgrl[Δoption]...
```

option: Option name
file: File name
[ ]: Can be omitted
...: Pattern in preceding [ ] can be repeated
{ }: Select from items delimited by the pipe symbol ("|")
Δ: One or more spaces
[, ...]: The preceding pattern can be repeated by delimiting each with a comma.
[: ...]: The preceding pattern can be repeated by delimiting each with a colon.
string := *A*: *string* is replaced with *A*.
string := *A* | *B* | *C*: *string* is replaced with any one of *A*, *B*, or *C*.

The following points should be noted when entering a command.

- The specification formats of options depend on the command that is used.
  See "2.5.1 Compile options", "2.5.2 951Assemble options" and "2.5.3 Link options" for cautions about options of each command.
- A file name supported by the OS can be specified.
  To specify a file name, specify a relative path or an absolute path beginning with a drive name.
  "-" cannot be also used at the beginning of a file name because it is regarded as the option specification.
  "(" and ")" cannot be also used for a file name because they are regarded as the part of link options.
  In addition, there are cautions on using characters in file names and path names of subcommand files used for internal processing.
  Also refer to "2.4.2 Subcommand file usage".
- The length that can be specified for a file name depends on the OS (up to 259 characters in Windows).

- Alphabetical file names are not case sensitive in Windows.
- Two or more files can be specified as input.
  Files which have different types (C source file and assembly source file or object file, and the like) can be mixed.
  Note that two or more files having the same source file name except for the extension cannot be specified (even when they are stored in separate folders).
  In this case, even if there is an error in one file, processing of the remaining files will continue if processing is possible.
  The generated object files are not deleted after linking.

(2) Example of operations
The examples of operations on the command line are shown below.

Remark See "2.5 Option" for details about each option.

(a) Performing compilation, assembly, and linking by one command
C source file "file1.c" is compiled by ccrl, and then assembly source file "file1.asm" is generated.
Next, assembly source file "file1.asm" and "file2.asm" are assembled by asrl, and then object file "file1.obj" and "file2.obj" are generated.
The assemble list files are output to the current folder.
Finally, object file "file1.obj", "file2.obj", and "file3.obj" are linked by rlink, and then link map file "sample.map" and load module file "sample.abs" are generated.

```
>ccrl file1.c file2.asm file3.obj -asmopt=-prn_path -lnkopt=-list -osample.abs -
cpu=S2 -dev=dr5f100pj.dvf
```

Remark In the ccrl command line, use the -asmopt option to specify an option dedicated to asrl; to specify an option dedicated to rlink, use the -lnkopt option.

(b) Performing compilation and assembly by one command, and linking separately
C source file "file1.c" is compiled by ccrl, and then assembly source file "file1.asm" is generated.
Next, assembly source file "file1.asm" and "file2.asm" are assembled by asrl, and then object file "file1.obj" and "file2.obj" are generated.
The assemble list files are output to the current folder.

```
>ccrl -c file1.c file2.asm -asmopt=-prn_path -cpu=S2 -dev=dr5f100pj.dvf
```

Remark In the ccrl command line, use the -asmopt option to specify an option dedicated to asrl.

Object file "file1.obj", "file2.obj", and "file3.obj" are linked by rlink, and then link map file "sample.map" and load module file "sample.abs" are generated.

```
>rlink file1.obj file2.obj file3.obj -output=sample.abs -list
```

(c) Performing compilation, assembly, and linking separately
C source file "file1.c" is compiled by ccrl, and then assembly source file "file1.asm" is generated.

```
>ccrl -S file1.c -cpu=S2 -dev=dr5f100pj.dvf
```

Assembly source file "file1.asm" and "file2.asm" are assembled by asrl, and then object file "file1.obj" and "file2.obj" are generated.
Assemble list files are also output.

```
>asrl file1.asm -prn_path -cpu=S2 -dev=dr5f100pj.dvf
>asrl file2.asm -prn_path -cpu=S2 -dev=dr5f100pj.dvf
```

Object file "file1.obj", "file2.obj", and "file3.obj" are linked by rlink, and then link map file "sample.map" and load module file "sample.abs" are generated.

```
>rlink file1.obj file2.obj file3.obj -output=sample.abs -list
```

(d) Generating a standard library
The library generator (lbgrl) is started to generate a standard library.

```
>lbgrl[Δoption]...
```

2.4.2 Subcommand file usage

A subcommand file is a file that options and file names specified for a ccrl, asrl, rlink command are described.
The command treats the contents of a subcommand file as if they were command-line arguments.
Use a subcommand file when the arguments will not fit on the command line, or when same options are specified repeatedly each time the command is executed.

(1) Using a subcommand file for the compiler and assembler

(a) Cautions about description of a subcommand file

- The arguments to be specified can be coded over several lines.
  However, you cannot start a new line within the name of the option specification or file.
- When the subcommand option is specified in a subcommand file, the same file name as the current subcommand file cannot be specified in the subcommand option.
- The character code contents of a subcommand file cannot be specified by using the -character_set option.
  If you use characters other than ASCII in the subcommand file, use the UTF-8 file with BOM.
- The following characters are treated as special characters.
  Special characters written in a subcommand file are deleted from the parameter string passed to the ccrl command, and the ccrl command is executed with that string.

" (double quotation mark)	The character string until the next double quotation mark is treated as a contiguous character string.
# (sharp)	If this is specified at the beginning of a line, the characters on that line before the end of the line are interpreted as a comment.
^ (circumflex)	The character immediately following this is not treated as a special character.

(b) Example of subcommand file specification
Create subcommand file "sub.txt" using an editor.

```
-cpu=S2
-dev=dr5f100pj.dvf
-c
-D test
-I dir
-Osize
```

Specify sub.txt by subcommand file specification option "-subccomand" on the command line.

```
>ccrl -subccomand=sub.txt -ofile.obj file.c
```

The command line is expanded as follows.

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -c -D test -I dir -Osize -ofile.obj file.c
```

(2) Using a subcommand file for the optimizing linker

(a) Cautions about description of a subcommand file

- The leading hyphen ("-") on option names can be omitted.
- A space can be used in place of the equals sign ("=") as the delimiter between the option and parameter.
- A character string enclosed by double quotation marks (") can be specified as a file name or a path name.
- A subcommand file should be written in the character code specified in the system locale.
- Specify one option per one line.
  If the command line cannot fit on a single line, you can use the ampersand ("&") to span multiple lines.
- The subcommand option cannot be specified in a subcommand file.
  [V1.03 or earlier]

- When the subcommand option is specified in a subcommand file, the same file name as the current subcommand file cannot be specified in the subcommand option.
  [V1.04 or later]
- The following characters are treated as special characters.
  These special characters themselves are not included in the command line of the rlink command and deleted.

& (and)	The following line will be treated as a continuation.
; (semicolon)	The characters on that line before the end of the line are interpreted as a comment.

(b) Example of subcommand file specification
Create subcommand file "sub.txt" using an editor.

```
input file2.obj file3.obj       ; This is a comment.
library lib1.lib, &             ; This is a line continued.
lib2.lib
```

Specify sub.txt by subcommand file specification option "-subcommand" on the command line.

```
>rlink file1.obj -subcommand=sub.txt file4.obj
```

The command line is expanded as follows.

```
>rlink file1.obj file2.obj file3.obj -library=lib1.lib,lib2.lib file4.obj
```

2.5 Option

This section explains ccrl options for each phase.
Compile phase -> See "2.5.1 Compile options"
Assemble phase -> See "2.5.2 951Assemble options"
Link phase -> See "2.5.3 Link options"
Library generation phase -> See "2.5.4 Library generator options [V1.13.00 or later]"

2.5.1 Compile options

This section explains options for the compile phase.

Caution about options are shown below.

- Uppercase characters and lowercase characters are distinguished for options.
- When numerical values are specified as parameters, decimal or hexadecimal numbers which starts with "0x" ("0X") can be specified.
  Uppercase characters and lowercase characters are not distinguished for the alphabet of hexadecimal numbers.
- When a file name is specified as a parameter, it can include the path (absolute path or relative path).
  When a file name without the path or a relative path is specified, the reference point of the path is the current folder.
- When a parameter includes a space (such as a path name), enclose the parameter in a pair of double quotation marks (").

The types and explanations for options are shown below.
An option with the description of [Professional Edition only] can be used only in the Professional Edition.

Table 2.2 Compile Options

Classification	Option	Description
Version/help display specification	-V	This option displays the version information of ccrl.
	-help	This option displays the descriptions of ccrl options.
Output file specification	-o	This option specifies the output file name.
	-obj_path	This option specifies the folder to save an object file generated during compilation.
	-asm_path	This option specifies the folder to save an assembly source file generated during compilation.
	-prep_path	This option specifies the folder to save the preprocessed C source file.
Source debugging control	-g	This option outputs information for source debugging.
	-g_line [V1.02 or later]	This option enhances information for source debugging at optimization.
Device specification relation	-cpu	This option specifies the type of the CPU core.
	-use_mda	This option specifies whether to allow generation of a code using the division/multiplication and multiply-accumulate unit.
	-use_mach [V1.11.00 or later]	This option supports generation of the multiply-accumulate instruction MACHU/MACH provided by the S3 core.
Language standard specification	-lang [V1.06 or later]	This option specifies the language standard of C source files and C++ source files.
Processing interrupt specification	-P	This option is used to execute only preprocessing for the input C source file.
	-S	This option does not execute processing after assembling.
	-c	This option does not execute processing after linking.

Classification	Option	Description
Preprocessor control	-D	This option defines preprocessor macros.
	-U	This option deletes the definition of the preprocessor macro by the -D option.
	-I	This option specifies the folder to search header files.
	-preinclude	This option specifies the file that is included at the top of the compilation unit.
	-preprocess	This option controls outputting the result of preprocessing.
Memory model	-memory_model	This option specifies the type of the memory model when compiling.
	-far_rom	This option sets the default near/far attribute of ROM data to far.
Optimization	-O	This option specifies the optimization level or the details of each optimization items.
	-goptimize	This option generates the information for inter-module optimization.
Error output control	-no_warning_num	This option suppresses outputting warning messages of the specified number.
	-change_message [V1.06 or later]	This option changes the message level.
	-error_file	This option outputs all compiler error messages together to a file.
Additional information output	-cref	This option outputs the static analysis information file.
	-pass_source	This option outputs a C source program as a comment to the assembly source file.

Classification	Option	Description
Code generation changing	-dbl_size	This option changes the interpretation of the double and long double types.
	-signed_char	This option specifies that a char type without a signed or unsigned specifier is handled as a signed type.
	-signed_bitfield	This option specifies that a bit field of a type without a signed or unsigned specifier is handled as a signed type.
	-switch	This option specifies the format in which the code of switch statements is to be output.
	-volatile	External variables and the variables specified with "#pragma address" are handled as if they were volatile-declared.
	-merge_string	This option merges string literals.
	-pack	This option sets 1 as the number of alignment for a structure member.
	-stuff [V1.10 or later]	This option allocates variables to sections separated according to the number of alignment.
	-stack_protector/ -stack_protector_all [Professional Edition only] [V1.02 or later]	This option generates a code for detection of stack smashing.
	-control_flow_integrity [Professional Edition only] [V1.06 or later]	This option generates code for the detection of illegal indirect function calls.
	-insert_nop_with_label [V1.05 or later]	This option inserts a local label and nop instruction.
Extensions	-strict_std [V1.06 or later] / -ansi [V1.05 or earlier]	The C source program is processed in strict compliance with the language standard which is specified with the -lang option.
	-refs_without_declaration	An error is generated upon reference of an undeclared function or a function that does not have a prototype declaration.
	-large_variable	This option sets the maximum size of a variable to 0xffff bytes.
	-nest_comment	This option enables nesting of /* */ comments.
	-character_set	This option specifies the Japanese/Chinese character code.
MISRA check	-misra2004 [Professional Edition only]	This option checks source code against the MISRA-C: 2004 rules.
	-misra2012 [Professional Edition only] [V1.02 or later]	This option checks source code against the MISRA-C: 2012 rules.
	-ignore_files_misra [Professional Edition only]	This option specifies files that will not be checked against the MISRA-C: 2004 rules or MISRA-C: 2012 rules.
	-check_language_extension [Professional Edition only]	This option enables the source-code checking of the MISRA-C: 2004 rules or MISRA-C: 2012 rules, which are partially suppressed by the extended language specifications.
	-misra_intermodule [Professional Edition only] [V1.08 or later]	This option checks source code in multiple files against the MISRA-C:2012 rules.

Classification	Option	Description
Subcommand file specification	-subcommand	This option specifies a subcommand file.
Assembler and linker control	-asmopt	This option specifies assemble options.
	-lnkopt	This option specifies link options.
	-asmcmd	This option specifies the use of a subcommand file to specify the assemble options to be passed to the assembler.
	-lnkcmd	This option specifies the use of a subcommand file to specify the link options to be passed to the optimizing linker.
	-dev	This option specifies the device file that the assembler and optimizing linker use.
Compiler transition support	-convert_cc	This option supports transition of programs written for other compilers.
	-unaligned_pointer_for_ca78k0r [V1.06 or later]	Indirect references by pointers are accessed in 1-byte units.

Version/help display specification

The version/help display specification options are as follows.

- -V
- -help

-V

This option displays the version information of ccrl.

[Specification format]

```
-V
```

- Interpretation when omitted
  Compilation is performed without displaying the version information of ccrl.

[Detailed description]

- This option outputs the version information of ccrl to the standard error output.
  It does not execute compilation.

[Example of use]

- To output the version information of ccrl to the standard error output, describe as:

```
>ccrl -V
```

-help

This option displays the descriptions of ccrl options.

[Specification format]

```
-help
```

- Interpretation when omitted
  The descriptions of ccrl options are not displayed.

[Detailed description]

- This option outputs the descriptions of ccrl options to the standard error output.
  It does not execute compilation.

[Example of use]

- To output the descriptions of ccrl options to the standard error output, describe as:

```
>ccrl -help
```

Output file specification

The output file specification options are as follows.

- -o
- -obj_path
- -asm_path
- -prep_path

-o

This option specifies the output file name.

[Specification format]

```
-o∆file
```

- Interpretation when omitted
  The output file name differs depending on the specificated option.
  The file is output to the current folder when the output folder is not specified.
  - When the -P option is specified
    The output file name will be the input file name with the extension replaced by ".i".
  - When the -S option is specified
    The output assembly source file name will be the source file name with the extension replaced by ".asm".
  - When the -c option is specified
    The output object file name will be the source file name with the extension replaced by ".obj".
  - Other than above
    The output load module file name will be the first input file name with the extension replaced by ".abs".

[Detailed description]

- This option specifies the output file name as *file*.
- If *file* already exists, it will be overwritten.
- This option is valid when processing is interrupted by specifying the -P, -S, or -c option.
  - If this option is specified with the -P option
    It is assumed that is the name of the file containing the results of preprocessing performed on the input file has been specified as *file*.
  - If this option is specified with the -S option
    It is assumed that an assembly source file name has been specified as *file*.
  - If this option is specified with the -c option
    It is assumed that an object file name has been specified as *file*.
  - Other than above
    It is assumed that the output file name to be set in the -output option for the optimizing linker is specified as *file*.
    When *file* has no extension, the output file name depends on the -form option for the optimizing linker.
    See the description of "Link options" for detail.
- An error will occur if two or more files are output.
- An error will occur if *file* is omitted.

[Example of use]

- To output the load module file with "sample.abs" as the file name, describe as:

```
>ccrl -o sample.abs -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-obj_path

This option specifies the folder to save an object file generated during compilation.

[Specification format]

```
-obj_path[=path]
```

- Interpretation when omitted
  The object file is saved under the source file name with the extension replaced by ".obj" to the current folder.

[Detailed description]

- This option specifies the folder to save an object file generated during compilation as *path*.
- If an existing folder is specified as *path*, the object file is saved under the source file name with the extension replaced by ".obj" to *path*.
  An error will occur if a nonexistent folder is specified.
- An existing file can be specified as path.
  If one object file is output, it will be saved with *path* as the file name.
  If two or more object files are output, an error will occur.
  An error will occur if a nonexistent file is specified.
- If "=*path*" is omitted, the object file is saved under the C source file name with the extension replaced by ".obj" to the current folder.
- If two or more files with the same name (even if they are in different folders) are specified as source files, then a warning is output, and an object file is only saved for the last source file to be specified.

[Example of use]

- To save the object file generated during compilation to folder "D:\sample", describe as:

```
>ccrl -obj_path=D:\sample -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-asm_path

This option specifies the folder to save an assembly source file generated during compilation.

[Specification format]

```
-asm_path[=path]
```

- Interpretation when omitted
  An assembly source file will not be output (except when specifying the -S option).

[Detailed description]

- This option specifies the folder to save an assembly source file generated during compilation as *path*.
- If an existing folder is specified as *path*, the assembly source file is saved under the C source file name with the extension replaced by ".asm" to *path*.
  An error will occur if a nonexistent folder is specified.
- An existing file can be specified as *path*.
  If one assembly source file is output, it will be saved with *path* as the file name.
  If two or more assembly source files are output, an error will occur.
  An error will occur if a nonexistent file is specified.
- If "=*path*" is omitted, the assembly source file is saved under the C source file name with the extension replaced by ".asm" to the current folder.
- If two or more files with the same name (even if they are in different folders) are specified as source files, then a warning is output, and an assembly source file is only saved for the last source file to be specified.

[Example of use]

- To save the assembly source file generated during compilation to folder "D:\sample", describe as:

```
>ccrl -asm_path=D:\sample -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-prep_path

This option specifies the folder to save the preprocessed C source file.

[Specification format]

```
-prep_path[=path]
```

- Interpretation when omitted
  A preprocessed C source file will not be output (except when specifying the -P option).

[Detailed description]

- This option specifies the folder to save a preprocessed C source file generated by specifying the -P option as *path*.
- If an existing folder is specified as *path*, the preprocessed C source file is saved under the C source file name with the extension replaced by ".i" to *path*.
  An error will occur if a nonexistent folder is specified.
- An existing file can be specified as *path*.
  If one preprocessed C source file is output, it will be saved with *path* as the file name.
  If two or more preprocessed C source files are output, an error will occur.
  An error will occur if a nonexistent file is specified.
- If "=*path*" is omitted, the preprocessed C source file is saved under the C source file name with the extension replaced by ".i".
- If two or more files with the same name (even if they are in different folders) are specified as source files, then a warning is output, and a preprocessed C source file is only saved for the last source file to be specified.

[Example of use]

- To save the preprocessed C source file to folder "D:\sample", describe as:

```
>ccrl -prep_path=D:\sample -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Source debugging control

The source debugging control options are as follows.

- -g
- -g_line [V1.02 or later]

-g

This option outputs information for source debugging.

[Specification format]

```
-g
```

- Interpretation when omitted
  Information for source debugging will not be output.

[Detailed description]

- This option outputs information for source debugging to the output file.
- Source debugging can be performed by specifying this option.
- When both this option and the optimization option are specified, the information output for source debugging is also affected.

[Example of use]

- To output information for source debugging to the output file, describe as:

```
>ccrl -g -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-g_line [V1.02 or later]

This option enhances information for source debugging at optimization.

[Specification format]

```
-g_line
```

- Interpretation when omitted
  This option does not enhance information for source debugging at optimization.

[Detailed description]

- This option is valid only when the -g option is specified simultaneously.
- This option enhances debugging information so that step execution in the source level can be conducted more precisely at debugging when optimization has been performed.
- The amount of debugging information may increase and cause step execution to slow down.

[Example of use]

- To enhance the information for source debugging in the output file and then output it, describe as:

```
>ccrl -g -g_line -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Device specification relation

The device specification relation options are as follows.

- -cpu
- -use_mda
- -use_mach [V1.11.00 or later]

-cpu

This option specifies the type of the CPU core.

[Specification format]

```
-cpu={S1|S2|S3}
  S1: RL78-S1 core
  S2: RL78-S2 core
  S3: RL78-S3 core
```

- Interpretation when omitted
  Cannot be omitted. An error will occur if no specification is made.
  However, no error will occur when the -V or -help option is specified.

[Detailed description]

- This option specifies the type of the CPU core.
- An error will occur if the string that cannot be specified is specified.
- An error will occur if the -cpu=S1 and -dbl_size=8 options are specified at the same time.
- An error will occur if the -cpu=S2 and -dbl_size=8 options are specified at the same time.
- This option only specifies the type of the CPU core and does not specify whether the arithmetic unit is implemented.
- Depending on this option setting, the interpretation differs when the -use_mda option is omitted.
  See "-use_mda" for detail.
- Depending on this option setting, the interpretation differs when the -memory_model option is omitted.
  See "-memory_model" for detail.

[Example of use]

- To generate a code for the RL78-S2 core specified as the CPU type, describe as:

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-use_mda

This option specifies whether to allow generation of a code using the division/multiplication and multiply-accumulate unit.

[Specification format]

```
-use_mda={not_use|mda}
  not_use: Suppresses generation of a code by the compiler using the division/multipli-
           cation and multiply-accumulate unit.
  mda: Allows generation of a code by the compiler using the division/multiplication
       and multiply-accumulate unit.
```

- Interpretation when omitted
  When the -cpu=S1 or -cpu=S3 option is specified, the code is the same as when the -use_mda=not_use option is specified.
  When the -cpu=S2 option is specified, the code is the same as when the -use_mda=mda option is specified.

[Detailed description]

- This option specifies whether to allow generation of a code by the compiler using the division/multiplication and multiply-accumulate unit.
- The following shows the ccrl operation when both this option and the -cpu option are specified.

		-use_mda=	
		not_use	mda
-cpu=	S1	Can be specified together	Compile error
	S2	Can be specified together	Can be specified together
	S3	Can be specified together	Compile error

[Caution]

- To control codes other than those generated by the compiler, isolation of the runtime library is also required. For details, see "7.1 Supplied Libraries".
- Do not specify -use_mda=mda for devices without division/multiplication and multiply-accumulate units.

[Example of use]

- To generate a code that uses the division/multiplication and multiply-accumulate unit, describe as:

```
>ccrl -use_mda=mda -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-use_mach [V1.11.00 or later]

This option supports generation of the multiply-accumulate instruction MACHU/MACH provided by the S3 core.

[Specification format]

```
-use_mach={not_use|mach}
  not_use  : Generates a code that does not use the instruction MACHU/MACH
  mach     : Generates a code that uses the instruction MACHU/MACH
```

- Interpretation when omitted
  The code is the same as when the -use_mach=not_use option is specified.

[Detailed description]

- This option specifies whether to use the multiply-accumulate instruction MACHU/MACH.
- If the -use_mach=mach option is specified when the -cpu=S1 or -cpu=S2 option is specified, a compilation error occurs.
- If the -use_mach=mach option is specified, the system register MACR used by the multiply-accumulation instruction MACHU/MACH will be guaranteed to have the same values before and after function calls and interrupts.

Language standard specification

The language standard specification option is as follows.

- -lang [V1.06 or later]

-lang [V1.06 or later]

This option specifies the language standard of C source files and C++ source files.

[Specification format]

```
-lang={c|c99} [V1.11 or earlier]
-lang={c|c99|cpp14} [V1.12 or later]
```

- Interpretation when omitted
  Compilation is executed in accord with the C90 standard.

[Detailed description]

This option specifies the language standard of the source file.

- When the -lang=c option is specified or the option is omitted, compilation is executed in accord with the C90 standard.
- When the -lang=c99 option is specified, compilation is executed in accord with the C99 standard.
- When the -lang=cpp14 option is specified, compilation is executed in accord with the C++14 standard. [V1.12 or later]
- If any option other than -lang=c, -lang=c99, or -lang=cpp14 is specified, an error occurs.

[Remark]

- If a C source file is specified for input when the -lang=cpp14 option is specified, a compile error occurs.
  [V1.12 or later]
- This compiler does not support the following language standards.
  - Some header files and standard library functions in the C90/C99 language standard
  - Complex number types in the C99 language standard
  - Variable-length arrays in the C99 language standard

Processing interrupt specification

The processing interrupt specification options are as follows.

- -P
- -S
- -c

-P

This option is used to execute only preprocessing for the input C source file.

[Specification format]

```
-P
```

- Interpretation when omitted
  Processing is continued after preprocessing.
  The preprocessed C source file are not output.

[Detailed description]

- This option is used to execute only preprocessing for the input C source file and output the results to a file.
- The output file name will be the input file name with the extension replaced by ".i".
- The output file name can be specified by specifying this option and the -o option.
- The contents of the output file can be controlled by specifying the -preprocess option.

[Example of use]

- To execute only preprocessing for the input C source file and output the results to file "main.i", describe as:

```
>ccrl -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-S

This option does not execute processing after assembling.

[Specification format]

```
-S
```

- Interpretation when omitted
  Processing is continued after assembling.

[Detailed description]

- This option does not execute processing after assembling.
- The assembly source file is output under the source file name with the extension replaced by ".asm".
- The output file name can be specified by specifying this option and the -o option.

[Example of use]

- To output assembly source file "main.asm" without executing any processing after the assembling, describe as:

```
>ccrl -S -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-c

This option does not execute processing after linking.

[Specification format]

```
-c
```

- Interpretation when omitted
  Processing is continued after linking.

[Detailed description]

- This option does not execute processing after linking.
- The object file is output under the source file name with the extension replaced by ".obj".
- The output file name can be specified by specifying this option and the -o option.

[Example of use]

- To output object file "main.obj" without executing any processing after the linking, describe as:

```
>ccrl -c -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Preprocessor control

The preprocessor control options are as follows.

- -D
- -U
- -I
- -preinclude
- -preprocess

-D

This option defines preprocessor macros.

[Specification format]

```
-D[∆]name[=def][,name[=def]]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option defines *name* as a preprocessor macro.
- This is equivalent to adding "#define *name def*" at the beginning of the source file.
- This option can be used to redefine C language macros that have been defined already: __STDC__, __LINE__, __FILE__, __DATE__, __TIME__, __RENESAS_VERSION__, __RL78__, __CCRL__, __CCRL, __RENESAS__ (except for -D __RL78__[=1], -D __CCRL__[=1], -D __CCRL[=1] and -D __RENESAS__[=1]).
  If any of these macros is redefined, a warning will be output.
- An error will occur if *name* is omitted.
- If "=*def*" is omitted, *def* is regarded as 1.
- This option can be specified more than once.
- If both this option and -U option are specified for the same preprocessor macro, the option specified last will be valid.

[Example of use]

- To define "sample=256" as a preprocessor macro, describe as:

```
>ccrl -D sample=256 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-U

This option deletes the definition of the preprocessor macro by the -D option.

[Specification format]

```
-U[∆]name[,name]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes the definition of the preprocessor macro by the -D option.
- This is equivalent to adding "#undef *name*" at the beginning of the source file.
- An error will occur if *name* is omitted.
- This option cannot delete the definition by describing "#define *name def*".
- This option can be specified more than once.
- If both this option and -D option are specified for the same preprocessor macro, the option specified last will be valid.

[Example of use]

- To delete the definition of preprocessor macro "test" by the -D option, describe as:

```
>ccrl -D TEST=XTEST -U TEST -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-I

This option specifies the folder to search header files (compile phase/assemble phase).

[Specification format]

```
-I[∆]path[,path]...
```

- Interpretation when omitted
  The header file is searched from only the folder that holds the source file and the standard header file folder (*version* folder\inc) (compile phase).
  The include file is searched from only the folder that holds the source file and the current folder (assemble phase).

[Detailed description]

- This option specifies the folder to search for header files which are loaded by preprocessor directive "#include" and include files which are loaded by the assembler's control instruction "$INCLUDE" as *path*.
  Header files are searched according to the following sequence.

  <1> Folder with source files (When files are specified by using " ")

  Remark When inclusion of a file is specified through #include, the folder that holds the file where the #include line is written is searched. When #include specifications are nested, folders are searched in the order from the innermost to the outermost level of the #include nest.

  <2> Path specified by the -I option (If multiple paths are specified, they are searched in the order in which they were specified on the command line (that is, from left to right).)

  <3> Standard header file folder

  The include files of the assembler are searched according to the following sequence.

  <1> Path specified by the -include option (If multiple paths are specified, they are searched in the order in which they were specified on the command line (that is, from left to right).)

  <2> Folder that holds the source fil

  <3> Current folder

- If *path* does not exist, a warning will be output.
- An error will occur if *path* is omitted.

[Example of use]

- To search header files from the current folder, folder D:\include, the standard folder in that order, describe as:

```
>ccrl -I D:\include -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-preinclude

This option specifies the file that is included at the top of the compilation unit.

[Specification format]

```
-preinclude=file[,file]...
```

- Interpretation when omitted
  It is assumed that the file that is included at the top of the compilation unit does not exist.

[Detailed description]

- This option specifies the file that is included at the top of the compilation unit as *file*.
- If the file specified as *file* is not found, an error will occur.
- This option starts searching from the folder that started the complier if the *file* is specified by its relative path.

[Example of use]

- To include file "sample.h" at the top of the compilation unit, describe as:

```
>ccrl -preinclude=sample.h -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-preprocess

This option controls outputting the result of preprocessing.

[Specification format]

```
-preprocess=string[,string]
```

- Interpretation when omitted
  The comments and line number information of the C source are not output to the preprocessed file.

[Detailed description]

- This option outputs the comments and line number information of the C source to the preprocessed file.
- This option is valid only when the -P option is specified.
  If the -P option is not specified, a warning is output and this option will be ignored.
- The items that can be specified as *string* are shown below.
  An error will occur if any other item is specified.

comment	Outputs the comments of the C source.
line	Outputs line number information.

<Format of line number information>

```
#line line-number "file-name"
```

- *line-number* is a decimal number, and the maximum value is the maximum number of unsigned int.
- In the full path of *file-name*, "\" is converted to "\\", and "" to '\"'.
  Other than printable characters (including spaces) are output as "*3-digit octal number*" ("\\%03o").
  Line feed characters are converted to "\\n".
- If an input source file contains the preprocessor directive '#*number* "*string*"' or '#line *number* "*string*"', then *number* is used as *line-number*, and *string* as *file-name*.

- An error will occur if *string* is omitted.
- It is output in the standard character encoding of the OS.

[Example of use]

- To output the comments and line number information of the C source to the preprocessed file, describe as:

```
>ccrl -preprocess=comment,line -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

The following example is equivalent to the example above.

```
>ccrl -preprocess=comment -preprocess=line -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Memory model

The memory model options are as follows.

- -memory_model
- -far_rom

-memory_model

This option specifies the type of the memory model when compiling.

[Specification format]

```
-memory_model={small|medium}
```

- Interpretation when omitted
  When the -cpu=S1 option is specified, small is assumed.
  When the -cpu=S2 or -cpu=S3 option is specified, medium is assumed.

[Detailed description]

- This option specifies the type of the memory model when compiling.
- The following shows the memory model that can be specified.

Type	Memory Model	Description
small	Small model	The default attribute of both variables and functions is set to near.
medium	Medium model	The default attribute of variables is set to near and that for functions is set to far.

- An error will occur if multiple types of the memory model are specified at the same time.
- See "2.6.6 Relationship with near and far" for the relationship with near and far.

[Example of use]

- To specify the small model as the type of the memory model when compiling, describe as:

```
>ccrl -memory_model=small -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-far_rom

This option sets the default near/far attribute of ROM data to far.

[Specification format]

```
-far_rom
```

- Interpretation when omitted
  The default near/far attribute of ROM data is determined by the -memory_model option setting.

[Detailed description]

- This option changes the near/far attribute for ROM data, which is specified by the memory model, to far.
- This option is not applied to the ROM data with the __near or __far keyword specified.
- When this option is specified and a pointer points to const data, the near/far attribute of this pointer is far. However, when a pointer points to non-const data, its attribute is near. Therefore, though in violation of the C90 and C99 standard, the pointer size depends on whether the pointer points to const data. This option should be used on this understanding.

```
// -far_rom is specified.
      char* ptr;    // The pointer size is 2 bytes. It points to the char with __near
attribute.
const char* c_ptr;  // The pointer size is 4 bytes. It points to the const char with
__far attribute.
```

- When a standard library function[Note] using a pointer to a const variable as a parameter is linked, it is replaced with a standard library function for a far pointer by "#if defined(__FAR_ROM__)" in the standard header.

  Note puts, perror, atof, atoff, strtod, strtof, atoi, atol, strtol, strtoul, bsearch, qsort, memcpy, memmove, memcmp, memchr, memset, strcpy, strncpy, strcat, strncat, strcmp, strncmp, strchr, strcspn, strpbrk, strrchr, strspn, strstr, strlen

[Example of use]

- To change the near/far attribute for ROM data, which is specified by the memory model, to far, describe as:

```
>ccrl -far_rom -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Optimization

The optimization options are as follows.

- -O
- -goptimize

-O

This option specifies the optimization level or the details of each optimization items.

[Specification format]

```
-O[level]
-Oitem[=value][,item[=value]]...
```

- Interpretation when omitted
  The -Odefault option is assumed.

[Detailed description]

- This option specifies the optimization level or the details of each optimization items.
- The following items can be specified for *level*.

size	Optimization with the object size precedence Regards reducing the ROM/RAM capacity as important and performs the maximum optimization that is effective for general programs.
speed	Optimization with the execution speed precedence Regards improving the execution speed as important and performs the maximum optimization that is effective for general programs.
default	Default Does optimization that is effective for both the object size and execution speed.
lite [V1.12 or later]	Partial optimization Performs partial optimization with marginal affect on the debug function.
nothing	Optimization with debugging precedence Regards debugging as important and suppresses all optimization.

Note that the items that can be specified for *level* vary depending on the version.

[V1.11 or earlier] All items other than -Olite can be specified for *level* for both the paid editions and free evaluation edition.

[V1.12 or later] All items can be specified for *level* for the paid editions and the free evaluation edition within the trial period.
Only -Olite or -Onothing can be specified for a free evaluation edition whose trial period has expired. If any other value is specified, a warning message is output, and then the value is changed to -Olite.

- The items that can be specified as *item* and *value* are shown below.
  An error will occur if any other item is specified.

Optimization Item (*item*)	Parameter (*value*)	Description
unroll	0 to 4294967295 (Integer value)	Loop expansion The loop statements (for, while, and do-while) are expanded. Use *value* to specify the maximum rate of increase in code size after loop expansion. A value of 0 set as *value* has the same meaning as a value of 1. If *value* is omitted, it is assumed that "2" has been specified. This item is valid when the -Osize, -Ospeed, or -Odefault option is specified.

Optimization Item (*item*)	Parameter (*value*)	Description
delete_static_func	on or off	Deleting unused static functions If *value* is omitted, it is assumed that "on" has been specified.
inline_level	0 to 3 (Integer value)	Inline expansion for functions *value* signifies the level of the expansion. 0: Suppresses all inline expansion including the function for which "#pragma inline" is specified. 1: Performs inline expansion for only a function for which "#pragma inline" is specified. 2: Distinguishes a function that is the target of expansion automatically and expands it. 3: Distinguishes the function that is the target of expansion automatically and expands it, while minimizing the increase in code size. However, if 1 to 3 is specified, the function that is specified by "#pragma inline" may not be expanded according to the content of the function and the status of compilation. If *value* is omitted, 2 is assumed if the -Osize, -Ospeed, or -Odefault option is specified, and 1 is assumed if the -Olite option is specified. If the -Osize, -Ospeed, or -Odefault option is specified, all items for *value* are valid. If the -Olite option is specified, 0 and 1 are valid. If the -Onothing option is specified, 0 is only valid.
inline_size	0 to 65535 (Integer value)	Size for inline expansion Specify the maximum increasing rate (%) of the code size up to which inline expansion is performed. When 100 is specified, functions are expanded inline until the code size increases by 100%. If *value* is omitted, it is assumed that 100 has been specified. This item is valid when the -Oinline_level=2 option is specified (including when the -Ospped option is specified).
pipeline [V1.03 or later]	on or off	Pipeline optimization If *value* is omitted, it is assumed that "on" has been specified. This item is valid when the -Osize, -Ospeed, or -Odefault option is specified.
tail_call	on or off	Replacement of a function call at the end of a function with br instruction If "on" is specified, then if there is a function call at the end of a function, and certain conditions are met, a br instruction will be generated for that call rather than a call instruction. The ret code will be removed, reducing the code size. However, some debug functions cannot be used. If *value* is omitted, it is assumed that "on" has been specified.

Optimization Item (*item*)	Parameter (*value*)	Description
merge_files	-	Merging of multiple files before compilation If this option is omitted, compilation is done in input file units without merging the files. Multiple C source files are merged then compiled, and output as a single file. The output file name is the specified output file name when -o is specified. When -o is not specified, the output file name becomes the file name according to the interpretation when -o is omitted for the C source file name that has been specified first. When there is only one input file or this option is specified simultaneously with -P, this option is invalid. When this option is specified simultaneously with -S or -c, an empty file is created and its file name is in accordance with the interpretation when -o is omitted for the C source file name that has been specified second or later. When this option is specified simultaneously with -Oinline_level, inline expansion is performed between files. When linking an object file created with this option specified, operation is not guaranteed if linker option -delete, -rename, or -replace is specified simultaneously.
intermodule	-	Global optimization execution The main optimization contents are the following. - Optimization using alias analysis between procedures - Propagation of constants, such as parameters and return values
whole_program	-	Optimization assuming that the whole program consists only of the input file If this option is omitted, it is not assumed that the whole program consists only of the target file for compilation. The compilation is performed assuming that the following conditions are met. Operation is not guaranteed if these conditions are not met. - The values and addresses of extern variables defined in the files to be compiled will not be modified or referenced from outside those files. - If a file to be compiled calls a function defined outside the files to be compiled, the called function will never call a function in the files to be compiled. If this option is specified, it is assumed that the -Ointermodule option is specified. If two or more C source files are input, it is assumed that the -Omerge_files option is specified.
alias	ansi or noansi	Optimization considering the type pointed to by pointers If this option is omitted, noansi is assumed.
same_code [V1.02 or later]	on or off	The multiple same instruction sequences that are found within a single section of the compilation unit are integrated into a function. If *value* is omitted, it is assumed that on is specified. This option is valid when the -Osize, -Ospeed, or -Odefault option is specified.

Optimization Item (*item*)	Parameter (*value*)	Description
branch_chaining [V1.10 or later]	on or off	Optimization by reducing the branch instruction code size This option uses a branch instruction whose code size is small. To use a branch instruction whose code size is small, a branch destination may be another branch instruction which shares the same destination, not a direct branch to the final destination. As a result, although this option reduces the code size, it also lowers the execution speed. Note that using this optimization without specifying the -g_line option may affect the behavior of single-step execution. If the *value* is omitted, it is assumed that on has been specified. This option is valid when the -Osize or -Odefault option is specified.
align [V1.10 or later]	on or off	Optimization by changing the alignment condition The number of generated instructions is decreased, the code size is reduced and the execution speed is increased by changing the variable alignment condition and then combining multiple accesses into one when, for example, accessing contiguous areas in a structure-type variable. As a result of changing the alignment condition, padding data is filled in and the amount of consumption may increase in the data storage area. If *value* is omitted, it is assumed that on has been specified. This option is valid when the -Osize, -Ospeed, or -Odefault option is specified. This option is invalid if the -stuff option is specified at the same time.

- If this option is specified more than once for the same *item*, the option specified last will be valid.
- According to the -O*level* setting, each -O*item* item is set to the value shown in the following.
  The -O*item* items not listed in the table are not affected by the -O*level* setting.
  Note that optimization through the -O*level* setting does not precisely match the optimization result obtained by specifying each -O*item* value separately. For example, when the -Odefault level is specified as -O*level* and then each -O*item* is separately set to match the corresponding item value for the -Osize level, the optimization result is not equivalent to that obtained when -Osize is specified from the beginning.

Optimization Item (*item*)	Optimization Level (*level*)				
	-Osize	-Ospeed	-Odefault	-Olite	-Onothing
unroll	1	2	1	-	-
delete_static_func	on	on	on	on	off
inline_level	3	2	3	1	0
inline_size	0	100	0	-	-
tail_call	on	on	on	off	off
pipeline	on	on	on	-	-
same_code	on	off	off	-	-
branch_chaining	on	-	on	-	-
align	on	on	off	-	-

- The interpretation when the -O option is partially or entirely omitted is as follows.
  - If specification of the -O option is omitted, it is assumed that the -Odefault option has been specified.
  - If the *level* parameter of the -O option is omitted, it is assumed that size has been specified.
  - If only *-Oitem* is specified (*-Olevel* is not specified), it is assumed that -Odefault has been specified for *-Olevel.*

[Example of use]

- To perform optimization with the object size precedence, describe as:

```
>ccrl -Osize -cpu=S2 -dev=dr5f100pj.dvf main.c
```

- Partial or entire omission of the -O option is interpreted as follows.

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c               # Same as -Odefault
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -O main.c            # Same as -Osize
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -Ounroll=8 main.c    # Same as -Odefault
                                                        -Ounroll=8
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -O -Ounroll=8 main.c # Same as -Osize -Ounroll=8
```

- To perform global execution, describe as
  - Optimization using alias analysis between procedures

```
> ccrl -cpu=S2 im1.c -Odelete_static_func=off,intermodule
```

<C source code>

```
extern long x[2];
extern int y[2];
static long func1(long *a, int *b) {
    *a=0;
    *b=1;
    return *a;
}
long func2(void) {
    return func1(&x[0], &y[1]);
}
```

<Output code>

```
_func1@1:
    .STACK _func1@1 = 6
    movw de, ax
    push bc
    pop hl
    clrw ax
    movw [de+0x02], ax
    movw [de], ax
    onew ax
    movw [hl], ax
    clrw bc         ; 0 is directly assigned because a and b point to different
    clrw ax         ; addresses.
    ret
```

  - Propagation of constants, such as parameters and return values

```
> ccrl -cpu=S2 im2.c -Oinline_level=1,intermodule
```

<C source code>

```
static __near int func(int x, int y, int z) {
    return z-x-y;
}
int func2(void) {
    return func(3,4,8);
}
```

<Output code>

```
    .SECTION .text,TEXT
_func@1:
    .STACK _func@1 = 4
    onew ax         ; 1(=8-3-4) is directly assigned.
    ret
    .SECTION .textf,TEXTF
_func2:
    .STACK _func2 = 4
    movw de, #0x0008
    movw bc, #0x0004
    movw ax, #0x0003
    br !_func@1
```

- To perform optimization considering the type pointed to by pointers, describe as:

```
> ccrl -cpu=S2 al1.c -Oalias=ansi
```

<C source code>

```
long x, n;
void func(short *ps)
{
    n = 1;
    *ps = 2;
    x = n;
}
```

<Output code>
As ps and n have different types, the value of n will not be affected by "ps = 2;". Therefore, the value used for assignment in "n = 1" is used again at (A).
(When "ps = 2;" changes the value of n, the result will be changed.)

```
_func:
    .STACK _func = 4
    movw de, ax
    clrw ax
    movw bc, ax
    movw !LOWW(_n+0x00002), ax
    onew ax
    movw hl, ax
    movw !LOWW(_n), ax
    onew ax
    incw ax
    movw [de], ax
    movw ax, bc     ; (A) The value used for assignment in n = 1 is used again.
    movw !LOWW(_x+0x00002), ax
    movw ax, hl     ; (A)
    movw !LOWW(_x), ax
    ret
```

[Caution]

When source-level debugging is performed using an optimized object code, take account of the following ramifications.

- A variable may not be read or written at the location where that variable is referenced in the source program because an expression has been transformed (copy propagation, common subexpression recognition, etc.) due to optimization.
- Step execution may not be performed according to the source program because statements have been optimized (code sharing, elimination, rearrangement, etc.).
  In addition, a breakpoint may not be set for a particular statement. For example, if a statement has been deleted, a breakpoint cannot be set at that statement.
- There is a possibility that the variable's available range (range in which the variable can be referenced in the program) or variable's location (register or memory location) is changed due to optimization (rearrangement of statements, register allocation, etc.) of statements or variables.
- Step execution for an inline-expanded statement is performed not at the inline-expanded part but within the inline expansion source function.
  Since calling of the inline-expanded function is deleted, a breakpoint cannot be set there.
- Since a great amount of memory is consumed to process debugging information at compilation, it may result in "out of memory".

-goptimize

This option generates the information for inter-module optimization.

[Specification format]

```
-goptimize
```

- Interpretation when omitted
  The information for inter-module optimization is not generated.

[Detailed description]

- This option generates the additional information for inter-module optimization in the output file.
- At linkage, inter-module optimization is applied to files for which this option has been specified.
  For details on inter-module optimization, see the description of the link option -Optimize.

[Example of use]

- To generate the information for inter-module optimization, describe as:

```
>ccrl -c -goptimize -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Error output control

The error control options are as follows.

- -no_warning_num
- -change_message [V1.06 or later]
- -error_file

-no_warning_num

This option suppresses outputting warning messages of the specified number.

[Specification format]

```
-no_warning_num={num|num1-num2}[, ...]
```

- Interpretation when omitted
  All warning messages are output.

[Detailed description]

- This option suppresses outputting warning messages of the specified number.
- Specify the error numbers as *num*, *num1*, and *num2*.
  If the error number that does not exist, it will be ignored.
- An error will occur if *num*, *num1*, or *num2* is omitted.
- If *num1-num2* is specified, it is assumed that error numbers within the range have been specified.
- If this option is specified more than once, all of the given specifications will be effective.
- The error number specified by this option is the rightmost 5 digits of the 7-digit number following the "W".
- This option does not control messages that have been changed to be conveyed as errors with the -change_message option.
- This option controls the following message numbers.
  - W0510000 to W0529999 and W0550000 to W0559999 [V1.05 or earlier]
  - W0510000 to W0559999 [V1.06 or later]

[Example of use]

- To suppress outputting warning message "W0511146" and "W0511147", describe as:

```
>ccrl -no_warning_num=11146,11147 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-change_message [V1.06 or later]

This option changes the message levels.

[Specification format]

```
-change_message=error[={num|num1-num2}[, ...]]
```

- Interpretation when omitted
  Message levels are not changed.

[Detailed description]

- This option changes the specified warning messages to error messages.
  This option controls message numbers W0510000 to W0549999.
- The rightmost 5 digits of the message number are specified for *num*, *num1*, and *num2*.
  If a message number that does not exist is specified, it will be ignored.
- If no message number is included, operation is as if all message numbers have been specified.
- If *num1-num2* is specified, it is assumed that message numbers within the range have been specified.
- If this option is specified more than once, all of the given specifications will be effective.

-error_file

This option outputs all error messages of the compiler to the specified file.

[Specification format]

```
-error_file=file
```

- Interpretation when omitted
  Error messages are output to only the standard error output.

[Detailed description]

- This option outputs error messages to the standard error output and file *file*.
- If *file* already exists, it will be overwritten.
- An error will occur if *file* is omitted.

[Example of use]

- To output error messages to the standard error output and file "err", describe as:

```
>ccrl -error_file=err -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Additional information output

The additional information output options are as follows.

- -cref
- -pass_source

-cref

This option outputs the static analysis information file.

[Specification format]

```
-cref=path
```

- Interpretation when omitted
  The static analysis information file is not output.

[Detailed description]

- This option specifies the location where the static analysis information file to be generated during compilation as *path*.
- If an existing folder is specified as *path*, the static analysis information file is saved under the C source file name with the extension replaced by ".cref" to *path*.
- If an existing file name is specified or a non-existing folder or file name is specified, the static analysis information file is output with *path* as the file name when one static analysis information file is output.
  If two or more static analysis information files are output, an error will occur.
- An error will occur if "=*path*" is omitted.
- If two or more files with the same name (even if they are in different folders) are specified as source files, then a warning is output, and a static analysis information file is only saved for the last source file to be specified.

[Example of use]

- To output the static analysis information file as file name "info.cref", describe as:

```
>ccrl -cref=info.cref -cpu=S2 -dev=dr5f100pj.dvf main.cs
```

-pass_source

This option outputs a C source program as a comment to the assembly source file.

[Specification format]

```
-pass_source
```

- Interpretation when omitted
  The C source program is not output as a comment to the assembly source file.

[Detailed description]

- This option outputs a C source program as a comment to the assembly source file.
- The output comments are for reference only and may not correspond exactly to the code.
  Additionally, non-executed lines may not be output as comments (e.g. type declarations and labels).
  For example, comments concerning global variables, local variables, function declarations, etc., may be output to incorrect positions.
  By specifying the optimization options, the code may be deleted and only the comment may remain.

[Example of use]

- To output a C source program as a comment to the assembly source file, describe as:

```
>ccrl -pass_source -S -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Code generation changing

The code generation changing options are as follows.

- -dbl_size
- -signed_char
- -signed_bitfield
- -switch
- -volatile
- -merge_string
- -pack
- -stuff [V1.10 or later]
- -stack_protector/-stack_protector_all [Professional Edition only] [V1.02 or later]
- -control_flow_integrity [Professional Edition only] [V1.06 or later]
- -insert_nop_with_label [V1.05 or later]

-dbl_size

This option changes the interpretation of the double and long double types.

[Specification format]

```
-dbl_size={4|8}
```

- Interpretation when omitted
  Both the double type and long double type are handled as the single-precision floating-point type (as when the -dbl_size=4 option is specified).

[Detailed description]

- This option changes the interpretation of the double and long double types.
- When 4 is specified for the parameter, the double type and the long double type are handled as the single-precision floating-point type.
- When 8 is specified for the parameter, the double type and the long double type are handled as the double-precision floating-point type.
- An error will occur if neither 4 nor 8 is specified as the parameter.
- An error will occur if the -cpu=S1 and -dbl_size=8 options are specified at the same time.
- An error will occur if the -cpu=S2 and -dbl_size=8 options are specified at the same time.
- When the -dbl_size=4 option is specified and a standard library function for the double type is called, it is replaced with the corresponding standard library for the float type.
- This option affects the predefined macros.

[Example of use]

- To regard both the double type and the long double type as the float type, describe as:

```
>ccrl -dbl_size=4 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-signed_char

This option specifies that a char type without a signed or unsigned specifier is handled as a signed type.

[Specification format]

```
-signed_char
```

- Interpretation when omitted
  A char type without a signed or unsigned specifier as an unsigned type.

[Detailed description]

- When integer promotion is applied to a char type without a signed or unsigned specifier, it is handled as a signed type.
- This option affects the predefined macros.
- This option does not have effect on bit fields.
  Use the -signed_bitfield option for bit fields.

[Example of use]

- To specify that a char type without a signed or unsigned specifier is handled as a signed type, describe as:

```
>ccrl -signed_char -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-signed_bitfield

This option specifies that a bit field of a type without a signed or unsigned specifier is handled as a signed type.

[Specification format]

```
-signed_bitfield
```

- Interpretation when omitted
  A bit field of a type without a signed or unsigned specifier as an unsigned type.

[Detailed description]

- This option specifies that a bit field of a type without a signed or unsigned specifier is handled as a signed type.
- This option affects the predefined macros.

[Example of use]

- To specify that a bit field of a type without a signed or unsigned specifier is handled as a signed type, describe as:

```
>ccrl -signed_bitfield -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-switch

This option specifies the format in which the code of switch statements is to be output.

[Specification format]

```
-switch={ifelse|binary|abs_table|rel_table}
```

- Interpretation when omitted
  ccrl selects the optimum output format for each switch statement.

[Detailed description]

- This option specifies the format in which the code of switch statements is to be output.
- The parameters that can be specified are shown below.
  An error will occur if any other parameter is specified.

ifelse	Outputs the code in a format in which the case labels are compared one by one. This item should be specified when there are not so many case statements.
binary	Outputs the code in the binary search format. Searches for a matching case statement by using a binary search algorithm. If this item is selected when many labels are used, any case statement can be found at almost the same speed.
abs_table rel_table	Uses the case branch table in the switch statement to output a code. A table indexed by the case value is referenced to obtain the location of each case label from the case value and a branch to the location is done. The branch speed is almost the same for all case labels. When the case values are not sequential, unused areas are generated in the table. If the difference between the maximum and minimum case values in a switch statement exceeds 8192, this option setting is ignored and processing is done as described in "Interpretation when omitted". When abs_table is specified, the absolute address of each case label location is stored in the table. When rel_table is specified, the relative distance from a branch instruction to each case label location is stored in the table. However, if a relative distance exceeds 64 Kbytes, a linkage error will occur. When a function including a switch statement is allocated to the near area, a code using an absolute addressing table is generated regardless of which parameter is specified.

- Specify the -far_rom option in a device without a mirror area.
- An error will occur if the parameter is omitted.

[Example of use]

- To output a code for the switch statement in the binary search format, describe as:

```
>ccrl -switch=binary -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-volatile

External variables and the variables specified with "#pragma address" are handled as if they were volatile-declared.

[Specification format]

```
-volatile
```

- Interpretation when omitted
  Only the volatile-qualified variables are handled as if they were volatile-declared.

[Detailed description]

- All external variables and the variables specified with #pragma address are handled as if they were volatile-declared. The number of times and order in which external variables and variables specified with #pragma address are accessed are kept unchanged from those written in the C source file.

[Example of use]

- To handle all external variables and the variables specified with #pragma address as if they were volatile-declared, describe as:

```
>ccrl -volatile -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-merge_string

This option allocates string literals to a single area.

[Specification format]

```
-merge_string
```

- Interpretation when omitted
  If the same string literals are included multiple times in the source file, each will be allocated to a separate area.

[Detailed description]

- When the same string literals exist in the source file, this option merges them and allocates to the one area.
- The same string literals are allocated to the same area, regardless of whether #pragma section is specified.

[Example of use]

- When the same string literals exist in the source file, to merge them and allocate to the one area, describe as:

```
>ccrl -merge_string -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-pack

This option performs packing of a structure.

[Specification format]

```
-pack
```

- Interpretation when omitted
  Packing of a structure is not performed.

[Detailed description]

- This option sets 1 as the number of alignment for a structure member.
- When this option is specified, members of a structure are not aligned according to its type, but code is generated with them packed to be aligned at a 1-byte boundary.
- Correct operation is not guaranteed if there is a mixture of C source files with this option specified and C source files without this option specified.
- Correct operation is not guaranteed if a structure, union, or address of those members whose alignment condition has been changed from two bytes to one byte by this option is passed as an argument of a standard library function.
- Correct operation is not guaranteed if the address of a structure or union member whose alignment condition has been changed from two bytes to one byte by this option is passed to a pointer whose type has two bytes as the alignment condition and indirect reference to the pointer is performed.

[Example of use]

- To perform packing of a structure, describe as:

```
>ccrl -pack -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-stuff [V1.10 or later]

This option allocates variables to sections separated according to the number of alignment.

[Specification format]

```
-stuff[=<variable-type>[,...]]
<variable-type> : { bss | data | const }
```

- Interpretation when omitted
  Variables are allocated without separating sections.

[Detailed description]

- This option allocates the variables belonging to the specified <*variable-type*> to sections separated according to the number of alignment.
- bss specifies uninitialized variables, data specifies initialized variables, and const specifies const variables.
- If <*variable-type*> is omitted, all types of variables are applicable.
- If this option is specified multiple times, all specified types of variables are applicable.
- If the same variable type is specified multiple times, the compiler handles this as one specification. For this, no warning is issued.
- If anything other than bss, data, and const is specified for <*variable-type*>, an error occurs.
- Variables are output to a section whose section name has <*number-of-alignment*>.
  However, if the number of alignment is 2, "_2" is not added to a section name.
  Examples:
  When the number of alignment of variables is 2: .bss
  When the number of alignment of variables is 1: .bss_1

[Example of use]

```
// near area
const char  __near c_n = 1;
const short __near s_n = 2;
const long  __near l_n = 3;
// far area
const char  __far c_f = 1;
const short __far s_f = 2;
const long  __far l_f = 3;
```

Default	-stuff specification

`.SECTION .const,CONST`
`_c_n:`
`.DB 0x01`
`.ALIGN 2`
`_s_n:`
`.DB2 0x0002`
`.ALIGN 2`
`_l_n:`
`.DB4 0x00000003`
`.SECTION .constf,CONSTF`
`_c_f:`
`.DB 0x01`
`.ALIGN 2`
`_s_f:`
`.DB2 0x0002`
`.ALIGN 2`
`_l_f:`
`.DB4 0x00000003`
	`.SECTION .const_1,CONST,align=1`
	`_c_n:`
	`.DB 0x01`
	
	`.SECTION .const,CONST`
	`.ALIGN 2`
	`_s_n:`
	`.DB2 0x0002`
	`.ALIGN 2`
	`_l_n:`
	`.DB4 0x00000003`
	
	`.SECTION .constf_1,CONSTF,align=1`
	`_c_f:`
	`.DB 0x01`
	
	`.SECTION .constf,CONSTF`
	`.ALIGN 2`
	`_s_f:`
	`.DB2 0x0002`
	`.ALIGN 2`
	`_l_f:`
	`.DB4 0x00000003`

[Remark]

- Each section name reflects the following options or specification in #pragma section:
  -memory_model, -far_rom

-stack_protector/-stack_protector_all [Professional Edition only] [V1.02 or later]

This option generates a code for detection of stack smashing.

[Specification format]

```
-stack_protector[=num]
-stack_protector_all[=num]
```

- Interpretation when omitted
  A code for detection of stack smashing is not generated.

[Detailed description]

- This option generates a code for detection of stack smashing at the entry and end of a function.
- A 2-byte area is allocated just before the local variable area (in the direction towards address 0xFFFFF) at the entry to a function, and the value specified by *num* is stored. After that, the 2-byte area in which *num* was stored is checked for smashing at the end of the function. If smashing has occurred, the __stack_chk_fail function is called.
- The __stack_chk_fail function needs to be created by the user. It should be defined as a function having no parameters or return values, it should be located in the far area, and the processing to be executed at stack smashing should be written.
- Do not define the function as static.
- When calling another function in the __stack_chk_fail function, note that stack smashing is not detected recursively in the function that was called.
- Specify an integer from 0 to 65535 for *num*.
- If *num* is omitted, the compiler automatically determines the integer value.
- If -stack_protector is specified, this option generates a code for detection of stack smashing for only functions having a structure, union, or array that exceeds eight bytes as a local variable.
- If -stack_protector_all is specified, this option generates a code for detection of stack smashing for all functions.
- If this option is used simultaneously with #pragma stack_protector, the specification by #pragma stack_protector becomes valid.
- A code for detection of stack smashing is not generated for the functions in which the following is specified.
  #pragma inline, __inline keyword, #pragma inline_asm, #pragma no_stack_protector, #pragma rtos_interrupt, or #pragma rtos_task

[Example of use]

- To generate a code for detection of stack smashing, describe as:

```
>ccrl -stack_protector=1000 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-control_flow_integrity [Professional Edition only] [V1.06 or later]

This option generates code for the detection of illegal indirect function calls.

[Specification format]

```
-control_flow_integrity
```

- Interpretation when omitted
  Code for the detection of illegal indirect function calls is not generated.

[Detailed description]

- This option generates code for the detection of illegal indirect function calls.
  When this option is specified, code for the following processing is generated in the C source program.
  (1) The __control_flow_integrity checking function is called with an indirect calling address as an argument immediately before indirect function calls.
  (2) Within the checking function, the address given as the argument is checked against a list of the addresses of functions (hereafter referred to as the function list) which may be indirectly called. If the list does not include the address, the __control_flow_chk_fail function will be called since this is regarded as an illegal indirect function call.
  The correctness of processing to change the flow of the program, such as through indirect function calls, is referred to as control flow integrity (CFI), and CFI techniques are used to verify this.
- A checking function is defined as follows and provided as library functions.
  Calling the checking function in the same way as normal functions is prohibited.
- The compiler automatically extracts the information on the functions which may be indirectly called from the C source program. The linker consolidates that information in creating the function list. For the linker to create a function list, the -CFI link option must be specified.
  For details, refer to section 2.5.3 Link options.
- The __control_flow_chk_fail function contains code for the processing which is to be executed when an illegal indirect function call is detected. The user must define this function.
  Note the following when defining the __control_flow_chk_fail function.
  - Specify void as the type of the return value and parameter, and allocate it in the far area.
  - Do not define the function as static.
  - Calling the __control_flow_chk_fail function in the same way as a normal function is prohibited.
  - The __control_flow_chk_fail function is not for the creation of code for detecting illegal indirect function calls.
  - In the __control_flow_chk_fail function, note that execution must not be returned to the checking function, for example, by calling abort() to terminate the program.

[Example]

- <C source code>

```
#include <stdlib.h>

int glb;

void __control_flow_chk_fail(void)
{
    abort();
}

void func1(void) // Added to the function list.
{
    ++glb;
}

void func2(void) // Not added to the function list.
{
    --glb;
}

void (*pf)(void) = func1;

void main(void)
{
    pf(); // Indirect call of the function func1.
    func2();
}
```

- <Output code>
  When -cpu=S2 -S -control_flow_integrity is specified for compilation

```
___control_flow_chk_fail:
    .STACK ___control_flow_chk_fail = 4
    br !!_abort
_func1:
    .STACK _func1 = 4
    incw !LOWW(_glb)
    ret
_func2:
    .STACK _func2 = 4
    decw !LOWW(_glb)
    ret
_main:
    .STACK _main = 8
    subw sp, #0x04
    movw de, !LOWW(_pf)
    movw ax, de
    movw [sp+0x02], ax
    mov a, !LOWW(_pf+0x00002)
    mov [sp+0x00], a
    call !!___control_flow_integrity ; Call the checking function.
    mov a, [sp+0x00]
    mov cs, a
    movw ax, [sp+0x02]
    movw hl, ax
    call hl ; Indirect call of the function func1.
    call $!_func2 ; Direct call of the function func2.
    addw sp, #0x04
    ret
    .SECTION .bss,BSS
    .ALIGN 2
_glb:
    .DS (2)
    .SECTION .data,DATA
    .ALIGN 2
_pf:
    .DB2 LOWW(_func1)
    .DB LOW(HIGHW(_func1))
    .DB 0x00
```

-insert_nop_with_label [V1.05 or later]

This option inserts a local label and nop instruction.

[Specification format]

```
-insert_nop_with_label=file,line,label
```

- Interpretation when omitted
  A local label and nop instruction are not inserted.

[Detailed description]

- This option inserts a local label and nop instruction at the specified location based on the information for source debugging.
- When this option is specified, the -g option also becomes valid.
- This function is assumed to be used via CS+ or e2studio and should not be used directly by the user.

Extensions

The extensions options are as follows.

- -strict_std [V1.06 or later] / -ansi [V1.05 or earlier]
- -refs_without_declaration
- -large_variable
- -nest_comment
- -character_set

-strict_std [V1.06 or later] / -ansi [V1.05 or earlier]

The C source program is processed in strict compliance with the language standard.

[Specification format]

```
-strict_std [V1.06 or later]
-ansi       [V1.05 or earlier]
```

- Interpretation when omitted
  Compatibility with the conventional C language specifications is conferred and processing continues after warning is output. With C90 specified, some of the specifications that were added in C99 are acceptable.

[Detailed description]

- This option selects processing of the C source program in strict compliance with the language standard which is specified with the -lang option, and errors or warnings are output for code that violates the standard.
- When this option is specified, the macro name "__STDC__" is defined as a macro with the value 1.
- Processing when compilation is executed in strict compliance with the language standard is as follows.
  - Compliance with C90
    - _Bool type
      An error will occur.
    - long long type
      An error will occur.
    - #line-number
      An error will occur.
      If this option is not specified, "#line-number" will be handled in the same way as "#line line-number".
    - Type conversion
      Type conversions such as the assignment of a function pointer to a void pointer will cause errors.
    - Binary constants
      An error will occur. [V1.06 or later]
  - Compliance with C99 [V1.06 or later]
    - #line-number
      An error will occur.
      If this option is not specified, "#line-number" will be handled in the same way as "#line line-number".
    - Type conversion
      Type conversions such as the assignment of a function pointer to a void pointer will cause errors.
    - Binary constants
      An error will occur.

-refs_without_declaration

When a function without a declaration or a function with a declaration in the old style (K&R) is called, an error will occur.

[Specification format]

```
-refs_without_declaration
```

- Interpretation when omitted
  No message is output when a function without a declaration or a function with a declaration in the old style (K&R) is called.

[Detailed description]

- When a function without a declaration or a function with a declaration in the old style (K&R) is called, an error will occur.

[Example of use]

- When a function without a declaration or a function with a declaration in the old style (K&R) is called, an error will occur, describe as:

```
>ccrl -refs_without_declaration -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-large_variable

This option sets the maximum size of a variable to 0xffff bytes.

[Specification format]

```
-large_variable
```

- Interpretation when omitted
  The maximum size of a variable is set to 0x7fff bytes.
  Declaration of a variable with a size larger than 0x7fff bytes will cause an error.

[Detailed description]

- This option changes the maximum size of a variable from 0x7fff bytes to 0xffff bytes.
- Declaration of a variable with a size larger than 0xffff bytes will cause an error.
- When this option is specified and if the result of pointer subtraction exceeds the range of values that can be represented in signed int, the value cannot be correctly expressed in ptrdiff_t (signed int). Therefore, when this option is specified, take special care regarding the result of pointer calculation.

[Example of use]

- To set the maximum size of a variable to 0xffff bytes, describe as:

```
>ccrl -large_variable -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-nest_comment

This option enables nesting of /* */ comments.

[Specification format]

```
-nest_comment
```

- Interpretation when omitted
  Nested /* */ comments cause a warning.

[Detailed description]

- This option enables nesting of /* */ comments.

```
/*
  /*
    Nest
  */
*/
```

[Example of use]

- To enable nesting of /* */ comments, describe as:

```
>ccrl -nest_comment -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-character_set

This option specifies the Japanese/Chinese character code.

[Specification format]

```
-character_set={none|sjis|euc_jp|utf8|big5|gbk}
```

- Interpretation when omitted
  In a Japanese OS, sjis is assumed as the parameter for this option. In other OS's, none is assumed.

[Detailed description]

- This option specifies the character code to be used for Japanese/Chinese comments and character strings in the input file.
- The parameters that can be specified are shown below.
  An error will occur if any other item is specified.
  Operation is not guaranteed if the specified character code differs from the character code of the input file.

none	Does not process the Japanese and Chinese character code
euc_jp	EUC (Japanese)
sjis	SJIS
utf8	UTF-8
big5	Traditional Chinese
gbk	Simplified Chinese

- An error will occur if the parameter is omitted.

[Example of use]

- To specify EUC as the character code to be used for Japanese comments and character strings in the input file, describe as:

```
>ccrl -character_set=euc_jp -cpu=S2 -dev=dr5f100pj.dvf main.c
```

MISRA check

The MISRA check options are as follows.

- -misra2004 [Professional Edition only]
- -misra2012 [Professional Edition only] [V1.02 or later]
- -ignore_files_misra [Professional Edition only]
- -check_language_extension [Professional Edition only]
- -misra_intermodule [Professional Edition only] [V1.08 or later]

-misra2004 [Professional Edition only]

This option checks source code against the MISRA-C:2004 rules.

[Specification format]

```
-misra2004=item[=value]
```

- Interpretation when omitted
  The source code is not checked against the MISRA-C: 2004 rules.

[Detailed description]

- This option checks source code against the MISRA-C:2004 rules.
  A message is output if the item specified for the check is *item*.
- The items that can be specified as *item* are shown below.
  An error will occur if any other item is specified.

Check Item (*item*)	Parameter (*value*)	Description
all	None	The source code is checked against all of the rules which are supported.
apply	*num*[,*num*]...	The source code is checked against the rules with the numbers specified by *num* among the rules which are supported.
ignore	*num*[,*num*]...	The source code is checked against the rules with the numbers that are not specified by *num* among the rules which are supported.
required	None	The source code is checked against the rules of the "required" type among the rules which are supported.
required_add	*num*[,*num*]...	The source code is checked against the rules of the "required" type and the rules with the numbers specified by *num* among the rules which are supported.
required_remove	*num*[,*num*]...	The source code is checked against the rules of the "required" type except for the rules with the numbers specified by *num* among the rules which are supported.
file		The source code is checked against the rules with the numbers described in specified file *file* among the rules which are supported. Specify one rule number per one line in the file.

- The items that can be specified as *num* are shown below.
  An error will occur if any other item is specified.

  2.2 2.3
  4.1 4.2
  5.2 5.3 5.4 5.5 5.6
  6.1 6.2 6.3 6.4 6.5
  7.1
  8.1 8.2 8.3 8.5 8.6 8.7 8.11 8.12
  9.1 9.2 9.3
  10.1 10.2 10.3 10.4 10.5 10.6
  11.1 11.2 11.3 11.4 11.5
  12.1 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12 12.13
  13.1 13.2 13.3 13.4
  14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9 14.10
  15.1 15.2 15.3 15.4 15.5
  16.1 16.3 16.5 16.6 16.9

17.5
18.1 18.4
19.3 19.6 19.7 19.8 19.11 19.13 19.14 19.15
20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12

- An error will occur if *item* is omitted.

[Example of use]

- To check the source code against MISRA-C:2004 rule number: 5.2, 5.3, and 5.4, describe as:

```
>ccrl -misra2004=apply=5.2,5.3,5.4 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[Caution]

- An error will occur when this option is specified in the Standard Edition of the compiler.
- The source code cannot be simultaneously checked against the MISRA-C: 2012 rules.
- If the -lang=c99 option is specified, this option will be invalid.

-misra2012 [Professional Edition only] [V1.02 or later]

This option checks source code against the MISRA-C:2012 rules.

[Specification format]

```
-misra2012=item[=value]
```

- Interpretation when omitted
  The source code is not checked against the MISRA-C: 2012 rules.

[Detailed description]

- This option checks source code against the MISRA-C:2012 rules.
  A message is output if the item specified for the check is *item*.
- The items that can be specified as *item* are shown below.
  An error will occur if any other item is specified.
  The source code is always checked against the rules of the "mandatory" type regardless of the following specification.

Check Item (*item*)	Parameter (*value*)	Description
all	None	The source code is checked against all of the rules which are supported.
apply	*num*[,*num*]...	The source code is checked against the rules with the numbers specified by *num* among the rules which are supported.
ignore	*num*[,*num*]...	The source code is checked against the rules with the numbers that are not specified by *num* among the rules which are supported.
required	None	The source code is checked against the rules of the "mandatory" and "required" types among the rules which are supported.
required_add	*num*[,*num*]...	The source code is checked against the rules of the "mandatory" and "required" types and the rules with the numbers specified by num among the rules which are supported.
required_remove	*num*[,*num*]...	The source code is checked against the rules of the "required" type except for the rules with the numbers specified by num among the rules which are supported.
file		The source code is checked against the rules with the numbers described in specified file *file* among the rules which are supported. Specify one rule number per one line in the file.

- The items that can be specified as *num* are shown below. [V1.09]
  An error will occur if any other item is specified.

  2.2 2.6 2.7
  3.1 3.2
  4.1 4.2
  5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9
  6.1 6.2
  7.1 7.2 7.3 7.4
  8.1 8.2 8.3 8.4 8.5 8.6 8.8 8.9 8.11 8.12 8.13 8.14
  9.1 9.2 9.3 9.4 9.5
  10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8
  11.1 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9
  12.1 12.2 12.3 12.4 12.5
  13.1 13.2 13.3 13.4 13.5 13.6
  14.2 14.3 14.4

15.1 15.2 15.3 15.4 15.5 15.6 15.7
16.1 16.2 16.3 16.4 16.5 16.6 16.7
17.1 17.3 17.4 17.5 17.6 17.7 17.8
18.4 18.5 18.7
19.2
20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12 20.13 20.14
21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 21.10 21.11 21.12 21.13 21.15 21.16

- An error will occur if *item* is omitted.

[Example of use]

- To check the source code against MISRA-C:2012 rule number: 5.2, 5.3 describe as:

```
>ccrl -misra2012=apply=5.2,5.3 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[Caution]

- An error will occur when this option is specified in the Standard Edition of the compiler.
- The source code cannot be simultaneously checked against the MISRA-C: 2004 rules.

-ignore_files_misra [Professional Edition only]

This option specifies files that will not be checked against the MISRA-C: 2004 rules or MISRA-C: 2012 rules.

[Specification format]

```
-ignore_files_misra=file[,file]...
```

- Interpretation when omitted
  All C source files are checked.

[Detailed description]

- This option does not check file *file* against the MISRA-C: 2004 rules or MISRA-C: 2012 rules.
- This option is valid only when the -misra2004 or -misra2012 option is specified.
  If the -misra2004 or -misra2012 option is not specified, a warning is output and this option will be ignored.

[Example of use]

- Not to check sample.h against the MISRA-C: 2004 rules, describe as:

```
>ccrl -misra2004=all -ignore_files_misra=sample.h -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[Caution]

- An error will occur when this option is specified in the Standard Edition of the compiler.

-check_language_extension [Professional Edition only]

This option enables the source-code checking of the MISRA-C:2004 rules or MISRA-C: 2012 rules, which are partially suppressed by the extended language specifications.

[Specification format]

```
-check_language_extension
```

- Interpretation when omitted
  The source-code checking of the MISRA-C:2004 rules or MISRA-C: 2012 rules is disabled, which are partially suppressed by the extended language specifications.

[Detailed description]

- This option enables the source-code checks of the MISRA-C:2004 rules or MISRA-C: 2012 rules in the following cases where they are suppressed by the unique language specifications extended from the C language standard.
  - When the function has no prototype declaration (rule 8.1) and #pragma interrupt is specified for it.
- This option is valid only when the -misra2004 or -misra2012 option is specified.
  If the -misra2004 or -misra2012 option is not specified, a warning is output and this option will be ignored.

[Example of use]

- To enable the source-code checking of the MISRA-C:2004 rules, which are partially suppressed by the extended language specifications, describe as:

```
>ccrl -misra2004=all -check_language_extension -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[Caution]

- An error will occur when this option is specified in the Standard Edition of the compiler.

-misra_intermodule [Professional Edition only] [V1.08 or later]

This option checks source code in multiple files against the MISRA-C:2012 rules.

[Specification format]

```
-misra_intermodule=file
```

- Interpretation when omitted
  None (checking of source code in multiple files against the MISRA-C:2012 rules is disabled)

[Detailed description]

- This option saves symbol information of multiple files in *file* and checks source code in these files against the MISRA-C:2012 rules. If file does not exist, a new file will be created. If *file* exists, symbol information will be added to the file.
- This option is only valid when the -misra2012 option is specified. A warning is output and this option will be ignored if the -misra2012 option is not specified.
- An error will occur if *file* is omitted.
- This option is applied to rules classified as "System" in the analysis scope of MISRA-C:2012. Source code will be checked against the following MISRA-C:2012 rules. [V1.08]
  5.1 5.6 5.7 5.8 5.9
  8.3 8.5 8.6

[Example of use]

- To check source code in multiple files a.c, b.c, and c.c against the MISRA-C:2012 rules, describe as:

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -misra2012=all -misra_intermodule=test.mi a.c b.c
c.c
```

[Caution]

- .{c|a|f} cannot be specified as the extension of *file*. If specified, an error will occur. Correct operation is not guaranteed if *file* overlaps with another input or output file.
- If there are many files to be checked and the symbol information to be stored in *file* is huge, the compilation speed gets slower.
- If any of the source files is modified after *file* was created, recompilation will update the information of *file*. If any of the source files is deleted or its file name is changed, delete *file* and recheck source code against the MISRA-C:2012 rules because the information of *file* cannot be updated.
- An error will occur if this option is specified in the Standard edition of the compiler.

[Remark]

- This option cannot correctly check the source code when files are compiled in parallel by using, for example, parallel builds. Specify this option without performing parallel compilation.

Subcommand file specification

The subcommand file specification option is as follows.

- -subcommand

-subcommand

This option specifies a subcommand file.

[Specification format]

```
-subcommand=file
```

- Interpretation when omitted
  Only the options and file names specified on the command line are recognized.

[Detailed description]

- This option handles *file* as a subcommand file.
- An error will occur if *file* does not exist.
- An error will occur if *file* is omitted.
- See "2.4.2 Subcommand file usage" for details about a subcommand file.

[Example of use]

- To handle "command.txt" as a subcommand file, describe as:

```
>ccrl -subcommand=command.txt -cpu=S2 -dev=dr5f100pj.dvf
```

Assembler and linker control

The assembler and linker control options are as follows.

- -asmopt
- -lnkopt
- -asmcmd
- -lnkcmd
- -dev

-asmopt

This option specifies assemble options.

[Specification format]

```
-asmopt=arg
```

- Interpretation when omitted
  Only the assemble options specified by the compilation driver are passed to the assembler.

[Detailed description]

- This option passes *arg* to the assembler as the assemble option.
- An error will occur if *arg* is omitted.

[Example of use]

- To pass the -prn_path option to the assembler, describe as:

```
>ccrl -c -asmopt=-prn_path -cpu=S2 -dev=dr5f100pj.dvf main.c
```

The -asmopt option specified in the above example has the same effect as the following example.

```
>ccrl -S -cpu=S2 -dev=dr5f100pj.dvf main.c
>asrl -prn_path -cpu=S2 -dev=dr5f100pj.dvf main.asm
```

-lnkopt

This option specifies link options.

[Specification format]

```
-lnkopt=arg
```

- Interpretation when omitted
  Only the link options specified by the compilation driver are passed to the optimizing linker.

[Detailed description]

- This option passes *arg* to the optimizing linker as the link option.
- An error will occur if *arg* is omitted.

[Example of use]

- To pass the -form=relocate option to the optimizing linker, describe as:

```
>ccrl -lnkopt=-form=relocate -cpu=S2 -dev=dr5f100pj.dvf main.c
```

The -lnkopt option specified in the above example has the same effect as the following example.

```
>ccrl -c -cpu=S2 -dev=dr5f100pj.dvf main.c
>rlink -form=relocate main.obj
```

-asmcmd

This option specifies the use of a subcommand file to specify the assemble options to be passed to the assembler.

[Specification format]

```
-asmcmd=file
```

- Interpretation when omitted
  Only the assemble options specified by the compilation driver are passed to the assembler.

[Detailed description]

- This option specifies the use of subcommand file *file* to specify the assemble options to be passed to the assembler.
- When this option is specified more than once, all subcommand files are valid.
- An error will occur if *file* is omitted.

[Example of use]

- To specify the use of subcommand file "command_asm.txt" to specify the assemble options to be passed to the assembler.

```
>ccrl -asmcmd=command_asm.txt -cpu=S2 -dev=dr5f100pj.dvf
```

-lnkcmd

This option specifies the use of a subcommand file to specify the link options to be passed to the optimizing linker.

[Specification format]

```
-lnkcmd=file
```

- Interpretation when omitted
  Only the link options specified by the compilation driver are passed to the optimizing linker.

[Detailed description]

- This option specifies the use of subcommand file *file* to specify the link options to be passed to the optimizing linker.
- When this option is specified more than once, all subcommand files are valid.
- An error will occur if *file* is omitted.

[Example of use]

- To specify the use of subcommand file "command_lnk.txt" to specify the link options to be passed to the optimizing linker.

```
>ccrl -lnkcmd=command_lnk.txt -cpu=S2 -dev=dr5f100pj.dvf
```

-dev

This option specifies the device file that the assembler and optimizing linker use.

[Specification format]

```
-dev=file
```

- Interpretation when omitted
  No device file is passed to the assembler or optimizing linker.

[Detailed description]

- This option specifies device file *file* that the assembler and optimizing linker use.
- If this option is omitted at compilation, an error may occur in the assembler or the optimizing linker.
- An error will occur if this option is specified more than once.
- An error will occur if *file* is omitted.

[Example of use]

- To specify device file "DR5F100PJ.DVF" that the assembler and optimizing linker use, describe as:

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c
```

Compiler transition support

The compiler transition support options are as follows.

- -convert_cc
- -unaligned_pointer_for_ca78k0r [V1.06 or later]

-convert_cc

This option supports transition of programs written for other compilers.

[Specification format]

```
-convert_cc={ca78k0r|nc30|iar}
```

- Interpretation when omitted
  The function for supporting transition of programs written for other compilers is disabled.

[Detailed description]

- This option converts expanded functions of another compiler into expanded functions of the CC-RL. Operations come into compliance with the CC-RL specifications.
- The same operations as the compiler before transition are not guaranteed for unspecified, undefined, and implementation-defined items in the ANSI C language.
- A compile error will occur when this option is specified for more than once.
- Correct operation is not guaranteed when linking objects with different compiler as a parameter of this option.
- The parameters that can be specified are shown below.
  A compile error will occur if any other parameter is specified.

Parameter	Description
ca78k0r	Enables the function for supporting transition of CA78K0R expanded language specifications.
nc30	Enables the function for supporting transition of NC30 expanded language specifications.
iar	Enables the function for supporting transition of ICCRL78 (IAR compiler) expanded language specifications.

- If the -lang=c99 option is specified, this option will be invalid.

Operations when -convert_cc=ca78k0r is specified are shown below.

- The __CNV_CA78K0R__ macro is enabled.
- A keyword following #pragma is recognized when it consists of only uppercase characters or only lowercase characters.
  A keyword consisting of both uppercase and lowercase characters is handled as an unknown keyword.
- The expanded language specifications are handled as follows:

Table 2.3 Operation When Transition Support Option is Specified (-convert_cc=ca78k0r)

Functions of ca78k0r	Functions in CC-RL	Operation When the Option is Specified
__callt, callt	__callt	When the -strict_std option is not specified, the callt keyword is replaced with __callt.
__callf, callf	None	Not supported. A syntax error will occur.
__sreg, sreg	__saddr	The __sreg keyword is replaced with __saddr. When the -strict_std option is not specified, the sreg keyword is replaced with __saddr.

Functions of ca78k0r	Functions in CC-RL	Operation When the Option is Specified
__leaf, norec, noauto	None	Not supported. A syntax error will occur.
__boolean, boolean, bit	None	When the -strict_std option is specified, the __boolean keyword is replaced with char. When the -strict_std option is not specified, the __boolean, boolean, or bit keyword is replaced with _Bool.
__interrupt __interrupt_brk	#pragma interrupt #pragma interrupt_brk	When the #pragma directive for the function qualified with the keyword is in the same file, the keyword is deleted. Otherwise, the keyword is replaced with the #pragma directive.
__asm #asm ~ #endasm	#pragma inline_asm	Not supported. __asm is handled as a normal function call. #asm and #endasm will generate a syntax error.
__rtos_interrupt	#pragma rtos_interrupt	When the #pragma directive for the function qualified with the keyword is in the same file, the keyword is deleted. Otherwise, the keyword is replaced with the #pragma directive.
__pascal	None	Not supported. A syntax error will occur.
__flash	None	Not supported. A syntax error will occur.
__flashf	None	Not supported. A syntax error will occur.
__directmap	#pragma address	The keyword is deleted and #pragma address is newly created. When __sreg, sreg or __saddr is specified additionally, a compile error will occur. When multiple variables are specified in the same address, an error will occur.
__temp	None	Not supported. A syntax error will occur.
__near, __far	__near, __far	The operation rules for the far pointer conform to the CC-RL specifications. The location for writing the __near or __far keyword in a function declaration or function pointer declaration conforms to the CC-RL specifications. When conforming to the CA78KOR specifications, a syntax error will occur. For the operation rules for the far pointer, see "Pointer operation" in "Specifying memory allocation area (__near /__far)".
__mxcall	None	Not supported. A syntax error will occur.
#pragma sfr	#include "iodefine.h"	The #pragma directive is ignored and a warning message is output. Reference to SFR including bit access is converted into reference to a symbolic constant defined in iodefine.h. Inclusion of iodefine.h must be specified manually.

Functions of ca78k0r	Functions in CC-RL	Operation When the Option is Specified
#pragma vect #pragma interrupt	#pragma interrupt #pragma interrupt_brk	The specifications are replaced with the CC-RL specifications. The vect keyword is replaced with interrupt. When the interrupt request name is BRK_I, the interrupt or vect keyword is replaced with interrupt_brk. If the directive includes stack switching, the specification is deleted and a warning message is output. The interrupt request name is converted into the address defined in iodefine.h. Inclusion of iodefine.h must be specified manually. If the C source file has only a #pragma directive and no function declaration or function definition, no vector table is generated and no error will occur at linkage. Only a single interrupt request name can be set in an interrupt handler.
#pragma rtos_interrupt	#pragma rtos_interrupt	The specifications are replaced with the CC-RL specifications. The interrupt request name is converted into the address defined in iodefine.h. Inclusion of iodefine.h must be specified manually. If the C source file has only a #pragma directive and no function declaration or function definition, no vector table is generated and no error will occur at linkage.
#pragma rtos_task	#pragma rtos_task	The specifications are replaced with the CC-RL specifications.
#pragma di #pragma ei	__DI __EI	Call to function DI or EI is replaced with call to __DI or __EI, respectively.
#pragma halt #pragma stop #pragma brk #pragma nop	__halt __stop __brk __nop	Call to function HALT, STOP, BRK, or NOP is replaced with call to __halt, __stop, __brk, or __nop, respectively.
#pragma section	#pragma section	The compiler output section name is replaced with a section name conforming to the CC-RL specifications. If the directive includes an address specification, the specification is deleted and a warning message is output. If it cannot be replaced with a section name conforming to the CC-RL specifications, the #pragma directive is deleted and a warning message is output. For the section names that can be written in the CC-RL, see "Changing compiler output section name (#pragma section)".
#pragma name	None	The #pragma directive is deleted and a warning message is output.
#pragma rot	__rolb, __rorb, __rolw, __rorw	Call to function rolb, rorb, rolw, or rorw is replaced with call to __rolb, __rorb, __rolw, or __rorw, respectively.
#pragma mul	__mulu, __mului, __mulsi	Call to function mulu, muluw, or mulsw is replaced with call to __mulu, __mului, or __mulsi, respectively.
#pragma div	__divui, __remui	Call to function divuw or moduw is replaced with call to __divui or __remui, respectively.
#pragma mac	__macui, __macsi	Call to function macuw or macsw is replaced with call to __macui or __macsi, respectively.

Functions of ca78k0r	Functions in CC-RL	Operation When the Option is Specified
#pragma bcd	None	The #pragma directive is deleted and a warning message is output.
#pragma opc	None	The #pragma directive is deleted and a warning message is output.
#pragma ext_func	None	The #pragma directive is deleted and a warning message is output.
#pragma inline	None	If a line feed follows the #pragma directive, the #pragma directive is deleted and a warning message is output. If a function name following the #pragma directive is in the same line, the #pragma directive is handled as #pragma inline (with different function) in the CC-RL specifications.
Binary constant	Binary constant	Handled as a binary constant without change.
__K0R__	__RL78__	The macro is enabled (decimal constant 1).
__K0R__SMALL__	__RL78_SMALL__	The macro is enabled (decimal constant 1) when small is specified with the -memory_model option or when S1 is specified with the -cpu option while the -memory_model option is not specified.
__K0R__MEDIUM__	__RL78_MEDIUM__	The macro is enabled (decimal constant 1) when medium is specified with the -memory_model option or when other than S1 is specified with the -cpu option while the -memory_model option is not specified.
__K0R__LARGE__	None	Not supported. Handled as a user-defined macro.
__CHAR_UNSIGNED__	__UCHAR	The macro is enabled (decimal constant 1) when the -signed_char option is not specified.
__RL78_1__	__RL78_S2__	The macro is enabled (decimal constant 1) when S2 is specified with the -cpu option.
__RL78_2__	__RL78_S3__	The macro is enabled (decimal constant 1) when S3 is specified with the -cpu option.
__RL78_3__	__RL78_S1__	The macro is enabled (decimal constant 1) when S1 is specified with the -cpu option.
__CA78K0R__	None	The macro is enabled (decimal constant 1).
CPU macro	None	Not supported. Handled as a user-defined macro.
Standard library function va_starttop	va_start	In stdarg.h, va_starttop is replaced with va_start.
Standard library functions toup, _toupper, tolow, _tolower, _putc, calloc, free, malloc, realloc, atexit, brk, sbrk, itoa, ltoa, ultoa, strbrk, strsbrk, stritoa, strltoa, strultoa, strcoll, strxfrm, matherr, _assertfail	None	Not supported. Handled as a normal function call.

Functions of ca78k0r	Functions in CC-RL	Operation When the Option is Specified
standard library functions Others	Standard library functions	Conforms to the CC-RL specifications. The location for writing the __near or __far keyword in a function declaration or function pointer declaration conforms to the CC-RL specifications. When conforming to the CA78KOR specifications, a syntax error will occur.
Standard library Macro	Standard library Macro	A macro with the same name as a macro defined in the header file of the CC-RL conforms to the CC-RL specifications. Other macros are not supported. They are handled as user-defined macros.

Operations when -convert_cc=nc30 is specified are shown below.

- The __CNV_NC30__ macro is enabled.
- The expanded language specifications are handled as follows:

Table 2.4 Operation When Transition Support Option is Specified (-convert_cc=nc30)

Functions of nc30	Functions in CC-RL	Operation When the Option is Specified
wchar_t type	None	In stddef.h, the wchar_t type is declared as the unsigned short type using typedef.
Decimal constant with no suffix or with suffix l or L int long int long long int	Decimal constant with no suffix or with suffix l or L int long int long long int	Conforms to the CC-RL specifications.
Binary constant	Binary constant	Handled as a binary constant without change. "_" can be written between numeric values. If written in any other location, a syntax error will occur.
Wide character string	Wide character string	When combining a character string constant and a wide character string constant, conforms to the CC-RL specifications.
Default parameter declaration of function	None	Not supported. A syntax error will occur.
near, far _near, _far	__near, __far	The keyword is replaced with __near or __far. The operation rules for the far pointer conform to the CC-RL specifications. For the operation rules for the far pointer, see "Pointer operation" in "Specifying memory allocation area (__near /__far)".
asm, _asm	#pragma inline_asm	Not supported. Handled as a normal function call.
inline, _inline	__inline	The keyword is replaced with __inline.
restrict	None	The keyword is deleted and a warning message is output.
_ext4mptr	None	The keyword is deleted and a warning message is output.
#pragma ROM	None	The #pragma directive is deleted and a warning message is output.

Functions of nc30	Functions in CC-RL	Operation When the Option is Specified
#pragma SECTION	#pragma section	The compiler output section name is replaced with a section name conforming to the CC-RL specifications. If the invalid section type is used, the #pragma directive is deleted and a warning message is output. If it cannot be replaced with a section name conforming to the CC-RL specifications, a compile error will occur.
#pragma STRUCT	None	The #pragma directive is deleted and a warning message is output.
#pragma EXT4MPTR	None	The #pragma directive is deleted and a warning message is output.
#pragma ADDRESS	#pragma address	Handled as #pragma address in the CC-RL specifications. If the numeric notation of the address differs from that in the CC-RL specifications, the #pragma directive is deleted and a warning message is output.
#pragma BITADDRESS	None	The #pragma directive is deleted and a warning message is output.
#pragma INTCALL	None	The #pragma directive is deleted and a warning message is output.
#pragma INTERRUPT	#pragma interrupt	Handled as #pragma interrupt in the CC-RL specifications. If written in a format that differs from that in the CC-RL specifications, the #pragma directive is deleted and a warning message is output.
#pragma PARAMETER	None	The #pragma directive is deleted and a warning message is output.
#pragma SPECIAL	#pragma callt	The function specified by the #pragma directive is handled as the callt function and a warning message is output. The calling number is ignored.
#pragma ALMHANDLER	None	The #pragma directive is deleted and a warning message is output.
#pragma CYCHANDLER	None	The #pragma directive is deleted and a warning message is output.
#pragma INTHANDLER #pragma HANDLER	None	The #pragma directive is deleted and a warning message is output.
#pragma TASK	None	The #pragma directive is deleted and a warning message is output.
#pragma __ASMMACRO	None	The #pragma directive is deleted and a warning message is output
#pragma ASM ~ ENDASM	None	The #pragma directive is deleted and a warning message is output.
#pragma JSRA	None	The #pragma directive is deleted and a warning message is output.
#pragma JSRW	None	The #pragma directive is deleted and a warning message is output.
#pragma PAGE	None	The #pragma directive is deleted and a warning message is output.

Functions of nc30	Functions in CC-RL	Operation When the Option is Specified
#pragma SBDATA	None	The #pragma directive is deleted and a warning message is output.
NC30	None	The macro is enabled (a space is defined).
M16C	None	The macro is enabled (a space is defined).
__R8C__	None	The macro is enabled (a space is defined).
__cplusplus	None	Handled as a user-defined macro.
Standard library functions clearerr, fgetc, getc, fgets, fread, fscanf, fputc, putc, fputs, fwrite, fflush, fprintf, vfprintf, ungetc, ferror, feof, calloc, free, malloc, realloc, mblen, mbstowcs, mbtowc, wcstombs, wctomb, strcoll, stricmp, strnicmp, strxfrm, bzero, bcopy, memicmp, localeconv, setlocale	None	Not supported. Handled as a normal function call. When an unsupported header file is included, a compile error will occur.
Standard library functions Others	Standard library functions	Conforms to the CC-RL specifications. The location for writing the __near or __far keyword in a function declaration or function pointer declaration conforms to the CC-RL specifications.
Standard library Macro	Standard library Macro	A macro with the same name as a macro defined in the header file of the CC-RL conforms to the CC-RL specifications. Other macros are not supported. They are handled as user-defined macros.

Operations when -convert_cc=iar is specified are shown below.

- The __CNV_IAR__ macro is enabled.
- The expanded language specifications are handled as follows:

Table 2.5 Operation When Transition Support Option is Specified (-convert_cc=iar)

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
wchar_t type	None	In stddef.h, the wchar_t type is declared as the unsigned short type using typedef.
Anonymous union of file scope	None	Not supported. A syntax error will occur.
__near, __far	__near, __far	The operation rules for the far pointer conform to the CC-RL specifications. For the operation rules for the far pointer, see "Pointer operation" in "Specifying memory allocation area (__near /__far)".
__near_func, __far_func	__near, __far	The keyword is replaced with __near or __far. The operation rules for the far pointer conform to the CC-RL specifications. For the operation rules for the far pointer, see "Pointer operation" in "Specifying memory allocation area (__near /__far)".

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
__interrupt	#pragma interrupt	Replaced with "#pragma interrupt <function name>".
__monitor	None	The keyword is deleted and a warning message is output.
__no_bit_access	None	The keyword is deleted and a warning message is output.
__no_init	None	The keyword is deleted and a warning message is output.
__intrinsic	None	Not supported. A syntax error will occur.
__noreturn	None	The keyword is deleted and a warning message is output.
__no_save	None	The keyword is deleted and a warning message is output.
__root	None	The keyword is deleted and a warning message is output.
__ro_placement	None	The keyword is deleted and a warning message is output.
__sfr	None	Not supported. A syntax error will occur.
__saddr	__saddr	Handled as the __saddr keyword without change.
@ operator	#pragma address	Not supported. A syntax error will occur.
__segment_begin	__sectop	Conversion is not performed and an error message is output.
__segment_end	__secend	Conversion is not performed and an error message is output.
__segment_size	None	An error message is output.
__ALIGNOF__	None	An error message is output.
static_assert	None	An error message is output.
__break	__brk	Replaced with __brk.
__disable_interrupt	__DI	Replaced with __DI.
__enable_interrupt	__EI	Replaced with __EI.
__get_interrupt_level	None	Handled as a normal function call.
__get_interrupt_state	None	Handled as a normal function call.
__mach	None	Handled as a normal function call.
__machu	None	Handled as a normal function call.
__no_operation	__nop	Replaced with __nop.
__set_interrupt_level	None	Handled as a normal function call.
__set_interrupt_state	None	Handled as a normal function call.
__stop	__stop	Handled as __stop without change.

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
#pragma vector	#pragma interrupt	Replaced with "#pragma interrupt <function name> (vect = address)". The function name should be a function name in the function declaration subsequent to #pragma vector, and the __interrupt keyword is deleted. If there is no subsequent __interrupt function, the #pragma declaration is deleted. If multiple interrupt request names are specified for an interrupt handler, the first interrupt request name is set, and a warning message will be output for the second and subsequent interrupt request names and they will be ignored.
#pragma bank	#pragma interrupt	Replaced with "#pragma interrupt <function name> (bank={RB0 | RB1 | RB2 |RB3})". The function name should be a function name in the function declaration subsequent to #pragma bank, and the __interrupt keyword is deleted. The register bank after transition has "RB" added to the beginning of the number specified by #pragma bank. If there is no subsequent __interrupt function, the #pragma declaration is deleted.
#pragma basic_template_matching	None	The #pragma directive is deleted and a warning message is output.
#pragma bitfields	None	The #pragma directive is deleted and a warning message is output.
#pragma constseg	#pragma section	The #pragma directive is deleted and a warning message is output.
#pragma data_alignment	None	The #pragma directive is deleted and a warning message is output.
#pragma dataseg	#pragma section	The #pragma directive is deleted and a warning message is output.
#pragma diag_default	None	The #pragma directive is valid.
#pragma diag_error	None	The #pragma directive is valid.
#pragma diag_remark	None	The #pragma directive is valid.
#pragma diag_suppress	None	The #pragma directive is valid.
#pragma diag_warning	None	The #pragma directive is valid.
#pragma error	None	The #pragma directive is deleted and a warning message is output.
#pragma include_alias	None	The #pragma directive is deleted and a warning message is output.

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
#pragma inline	#pragma inline / #pragma noinline	Replaced with #pragma inline when forced is specified and with #pragma noinline when never is specified. Note that even when forced is specified, inline expansion is not always performed. The target function should be a function in the function declaration subsequent to #pragma inline. When other than a function declaration follows, an error will occur. When no function declaration follows, the #pragma directive is deleted and a warning message is output. Only #pragma inline in the IAR format can be used. #pragma inline in the CC-RL format leads to a compile error.
#pragma language	None	The #pragma directive is deleted and a warning message is output.
#pragma location	#pragma address	Replaced with #pragma address when an absolute address is specified. The variable name used in #pragma address should be a variable name in the variable declaration subsequent to #pragma location. When no variable declaration follows, the #pragma directive is deleted and a warning message is output. Segment names are not supported. They will lead to a syntax error.
#pragma message	None	The #pragma directive is deleted and a warning message is output.
#pragma object_attribute	None	The #pragma directive is deleted and a warning message is output.
#pragma optimize	None	The #pragma directive is deleted and a warning message is output.
#pragma pack	None	This option selects the conversion of structure type variables as #pragma pack for CC-RL when the number for alignment is regarded as 1 and as #pragma unpack for CC-RL when the number for alignment is regarded as 2. This option ignores other specifications of alignment and parameters that are not the numbers of alignment.
#pragma __printf_args	None	The #pragma directive is valid.
#pragma required	None	The #pragma directive is deleted and a warning message is output.
#pragma rtmodel	None	The #pragma directive is deleted and a warning message is output.
#pragma __scanf_args	None	The #pragma directive is valid.
#pragma segment	None	The #pragma directive is deleted and a warning message is output.
#pragma section	None	Handled as #pragma section in the CC-RL. When written in a format different from that in the CC-RL specifications, the #pragma directive is deleted and a warning message is output.
#pragma STDC CX_LIMITED_RANGE	None	The #pragma directive is deleted and a warning message is output.

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
#pragma STDC FENV_ACCESS	None	The #pragma directive is deleted and a warning message is output.
#pragma STDC FP_CONTRACT	None	The #pragma directive is deleted and a warning message is output.
#pragma type_attribute	None	The #pragma directive is deleted and a warning message is output.
#pragma unroll	None	The #pragma directive is deleted and a warning message is output.
#warning	None	The #pragma directive is valid.
_Pragma()	None	Handled as a normal function call.
__CORE__	None	The macro is enabled. Becomes one of the following values according to the specification of the -cpu option. - __RL78_0__ (when S1 is specified by the -cpu option) - __RL78_1__ (when S2 is specified by the -cpu option) - __RL78_2__ (when S3 is specified by the -cpu option)
__RL78_0__	__RL78_S1__	The macro is enabled (value is 1).
__RL78_1__	__RL78_S2__	The macro is enabled (value is 2).
__RL78_2__	__RL78_S3__	The macro is enabled (value is 3).
__CODE_MODEL__	None	The macro is enabled. Becomes one of the following values according to the specification of the -memory_model option or -cpu option. - __CODE_MODEL_NEAR__ (when small is specified by the -memory_model option or when S1 is specified by the -cpu option while the -memory_model option is not specified) - __CODE_MODEL_FAR__ (when medium is specified by the -memory_model option or when other than S1 is specified by the -cpu option while the -memory_model option is not specified)
__CODE_MODEL_NEAR__	__RL78_SMALL__	The macro is enabled (value is 1).
__CODE_MODEL_FAR__	__RL78_MEDIUM__	The macro is enabled (value is 2).
__DATA_MODEL__	None	The macro is enabled. The value becomes __DATA_MODEL_NEAR__ regardless of the specification of the -cpu option.
__DATA_MODEL_NEAR__	__RL78_SMALL__	The macro is enabled (value is 1).
__DATA_MODEL_FAR__	None	The macro is enabled (value is 2).
__func__	None	The macro is enabled.
__FUNCTION__	None	The macro is enabled.
__PRETTY_FUNCTION__	None	The macro is enabled.
__IAR_SYSTEMS_ICC__	None	The macro is enabled (value is 8).
__ICCRL78__	None	The macro is enabled (value is 1).

Functions of iar	Functions in CC-RL	Operation When the Option is Specified
__BUILD_NUMBER__ __cplusplus __DOUBLE__ __embedded_cplusplus __LITTLE_ENDIAN__ __SUBVERSION__ __VER__	None	Handled as a user-defined macro.
Standard library functions fabsl acosl asinl atanl atan2l ceill cosl coshl expl floorl fmodl frexpl ldexpl logl log10l modfl powl sinl sinhl sqrtl tanl tanhl strtold	None	Replaced with the following function names. fabs acos asin atan atan2 ceil cos cosh exp floor fmod frexp ldexp log log10 modf pow sin sinh sqrt tan tanh strtod The location for writing the __near or __far keyword in a function declaration or function pointer declaration conforms to the CC-RL specifications.
Standard library functions supported by CC-RL	Standard library functions	Conforms to the CC-RL specifications. The location for writing the __near or __far keyword for a function or function pointer in its declaration conforms to the CC-RL specifications.
Standard library functions Others	None	Not supported. Handled as a normal function call. When an unsupported header file is included, a compile error will occur.
Standard library Macro	Standard library Macro	A macro with the same name as a macro defined in the header file of the CC-RL conforms to the CC-RL specifications. Other macros are not supported. They are handled as user-defined macros.

[Example of use]

- To enable the function for supporting transition of programs written for ca78k0r, describe as:

```
>ccrl -convert_cc=ca78k0r -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-unaligned_pointer_for_ca78k0r [V1.06 or later]

Indirect references by pointers are accessed in 1-byte units.

[Specification format]

```
-unaligned_pointer_for_ca78k0r
```

- Interpretation when omitted
  This option generates code for indirect reference with 2-byte access for types having a 2-byte alignment condition.

[Detailed description]

- The purpose of this option is to support the porting of code written for the CA78K0R compiler. This option is specified when the same function was used with the CA78K0R compiler. Specifying this option increases the size of the object code and decreases the speed of execution.
- If a pointer to a type having a 2-byte alignment condition and without the volatile qualifier may indicate an odd address, this option generates code that handles indirect reference with 1-byte access. When the type is specified with the volatile qualifier, code handles indirect reference with 2-byte access even if an odd address is indicated for a type having the 2-byte alignment condition.
- When structure packing is performed by CC-RL, only the members of the structure do not have the 2-byte alignment condition; other types such as int have the 2-byte alignment condition and pointer reference to those types also has the 2-byte alignment condition. Thus, if the pointer to a member of a packed structure is assigned to a pointer to a type which is not to be packed, normal operation is not guaranteed. When this option is specified, since indirect reference by pointers involves 1-byte access, indirect reference by pointers to the members of packed structures is possible.

2.5.2 951Assemble options

This section explains options for the assemble phase.

Caution about options are shown below.

- Uppercase characters and lowercase characters are distinguished for options.
- When numerical values are specified as parameters, hexadecimal numbers which starts with "0x" ("0X") or decimal numbers can be specified.
  Uppercase characters and lowercase characters are not distinguished for the alphabet of hexadecimal numbers.
- When a file name is specified as a parameter, it can include the path (absolute path or relative path).
  When a file name without the path or a relative path is specified, the reference point of the path is the current folder.
- When a parameter includes a space (such as a path name), enclose the parameter in a pair of double quotation marks (").
- When the -prn_path, -mirror_source, -mirror_region, -define, -undefine, -include, -base_number, -warning, or -no_warning option is specified for ccrl command, the -asmopt option must be used.
  The -include option can also be specified as the -I option for the ccrl command.

The types and explanations for options are shown below.

Table 2.6 Assemble Options

Classification	Option	Description
Version/help display specification	-V	This option displays the version information of asrl.
	-help	This option displays the descriptions of asrl options.
Output file specification	-output	This option specifies the output file name.
	-obj_path	This option specifies the folder to save an object file generated after assembling.
	-prn_path	This option specifies the folder to save the assemble list file.
Source debugging control	-debug	This option outputs information for source debugging.
Device specification control	-dev	This option specifies the target device file with the path.
	-cpu	This option specifies the type of the CPU core.
	-mirror_source	This option specifies the value to be set in the MAA register.
	-mirror_region	This option specifies the address range of the mirror destination area.
Optimization	-goptimize	This option generates the information for inter-module optimization.
Symbol definition specification	-define	This option defines assembler symbols.
	-undefine	This option deletes the assembler symbol definition by the -define option.
Include file reading path specification	-include	This option specifies the folder to search include files.
Input file control	-character_set	This option specifies the Japanese/Chinese character code.
	-base_number	This option specifies the notation of the radix for numeric constants.
Assembler transition support	-convert_asm	This option enables the assembler transition supporting function.

Classification	Option	Description
Error message file output specification	-error_file	This option outputs error messages to a file.
Warning message output control	-warning	This option outputs the specified warning message.
	-no_warning	This option suppresses outputting warning messages of the specified number.
Subcommand file specification	@	This option specifies a subcommand file.

Version/help display specification

The version/help display specification options are as follows.

- -V
- -help

-V

This option displays the version information of asrl.

[Specification format]

```
-V
```

- Interpretation when omitted
  Assembling is performed without displaying the version information of asrl.

[Detailed description]

This option outputs the version information of asrl to the standard error output.
It does not execute assembling.

[Example of use]

- To output the version information of asrl to the standard error output, describe as:

```
>asrl -V
```

-help

This option displays the descriptions of asrl options.

[Specification format]

```
-help
```

- Interpretation when omitted
  The descriptions of asrl options are not displayed.

[Detailed description]

- This option outputs the descriptions of asrl options to the standard error output.
  It does not execute assembling.

[Example of use]

- To output the descriptions of asrl options to the standard error output, describe as:

```
>asrl -help
```

Output file specification

The output file specification options are as follows.

- -output
- -obj_path
- -prn_path

-output

This option specifies the output file name.

[Specification format]

```
-output=file
```

- Interpretation when omitted
  The file is output to the current folder.
  The output object file name will be the source file name with the extension replaced by ".obj".

[Detailed description]

- This option specifies the object file name as *file*.
- If *file* already exists, it will be overwritten.
- Even when this option is specified, if an error occurs and assembly processing cannot be continued, no object file will be output.
- An error will occur if two or more files are output.
- An error will occur if *file* is omitted.

[Example of use]

- To output the object file with "sample.obj" as the file name, describe as:

```
>asrl -output=sample.obj -dev=dr5f100pj.dvf main.asm
```

-obj_path

This option specifies the folder to save an object file generated after assembling.

[Specification format]

```
-obj_path[=path]
```

- Interpretation when omitted
  The object file is saved under the source file name with the extension replaced by ".obj" to the current folder.

[Detailed description]

- This option specifies the folder to save an object file generated after assembling as *path*.
- If an existing folder is specified as *path*, the object file is saved under the source file name with the extension replaced by ".obj" to *path*.
  An error will occur if a nonexistent folder is specified.
- An existing file can be specified as *path*.
  If one object file is output, it will be saved with *path* as the file name.
  If two or more object files are output, an error will occur.
  An error will occur if a nonexistent file is specified.
- If "=*path*" is omitted, the object file is saved under the C source file name with the extension replaced by ".obj" to the current folder.
- If two or more files with the same name (even if they are in different folders) are specified as source files, then a warning is output, and an object file is only saved for the last source file in the command line.

[Example of use]

- To save the object file generated during assembling to folder "D:\sample", describe as:

```
>asrl -obj_path=D:\sample -dev=dr5f100pj.dvf main.asm
```

-prn_path

This option specifies the folder to save the assemble list file.

[Specification format]

```
-prn_path[=path]
```

- Interpretation when omitted
  An assemble list file will not be output.

[Detailed description]

- This option specifies the folder to save the assemble list file output during assembling as *path*.
- If an existing folder is specified as *path*, the assemble list file is saved to folder *path*.
  When the extension of the input file name is ".asm", ".s", or ".fsy", the name with the extension replaced with ".prn" is used for the assemble list file.
  For other extensions, the file name with extension ".prn" added after the existing extension is used.
  An error will occur if a nonexistent folder is specified.
- An existing file can be specified as *path*.
  The assemble list file is saved with *path* as the file name.
  An error will occur if a nonexistent file is specified.
- If "=*path*" is omitted, the assemble list file is saved to the current folder.
  When the extension of the input file name is ".asm", ".s", or ".fsy", the name with the extension replaced with ".prn" is used for the assemble list file.
  For other extensions, the file name with extension ".prn" added after the existing extension is used.

[Example of use]

- To save the assemble list file output during assembling to folder "D:\sample", describe as:

```
>asrl -prn_path=D:\sample -dev=dr5f100pj.dvf main.asm
```

Source debugging control

The source debugging control option is as follows.

- -debug

-debug

This option outputs information for source debugging.

[Specification format]

```
-debug
```

- Interpretation when omitted
  Information for source debugging will not be output.

[Detailed description]

- This option outputs information for source debugging to the output file.
- Source debugging will become enabled by specifying this option.

[Example of use]

- To output information for source debugging to the output file, describe as:

```
>asrl -debug -dev=dr5f100pj.dvf main.asm
```

Device specification control

The device specification control options are as follows.

- -dev
- -cpu
- -mirror_source
- -mirror_region

-dev

This option specifies the target device file with the path.

[Specification format]

```
-dev=[path\]file
```

- Interpretation when omitted
  When the -cpu option is specified, the specification of the CPU core type by the -cpu option becomes valid.
  When the -cpu option is not specified, an error will occur.

[Detailed description]

- This option specifies target device file *file* with path *path*.
- The information read from the specified device file is used and an object file that matches the settings in the device file is generated.
- An error will occur if the specified device file is not found.
- When both this option and the -cpu option are specified and if the CPU core type in the device file specified by this option differs from that specified in the -cpu option, an error will occur.

[Example of use]

- To specify device file "DR5F100PJ.DVF", describe as:

```
>asrl -dev=dr5f100pj.dvf main.asm
```

-cpu

This option specifies the type of the CPU core.

[Specification format]

```
-cpu={S1|S2|S3}
  S1: RL78-S1 core
  S2: RL78-S2 core
  S3: RL78-S3 core
```

- Interpretation when omitted
  When the -dev option is specified, the CPU core written in the device file specified by -dev option is used. When the -dev option is not specified, an error will occur.

[Detailed description]

- An object file that can be used in common for the devices implementing the specified CPU core is output.
- An error will occur if the string that cannot be specified is specified.
- When both this option and the -dev option are specified and if the CPU core type specified in this option differs from that in the device file specified by the -dev option, an error will occur.
  In other cases, the specified device file is used for processing.
- The following shows other options that can be specified together with the -dev or -cpu option.

Option	When -dev Option Is Specified	When -cpu Option Is Specified
-mirror_source	Can be specified	Can be specified[Note]
-mirror_region	Cannot be specified	Can be specified

Note The -cpu=S1 option and -mirror_source=1 option cannot be specified at the same time.

[Example of use]

- To generate a code for the RL78-S2 core specified as the CPU type, describe as:

```
>asrl -cpu=S2 main.c
```

-mirror_source

This option specifies the value to be set in the MAA register.

[Specification format]

```
-mirror_source={0|1|common}
```

- Interpretation when omitted
  It is the same result as when the -mirror_source=0 option is specified.

[Detailed description]

- This option specifies the value to be set in the MAA register.
- Specify 0 when the mirror source section is allocated to address 0x0xxxx, or specify 1 when the section is allocated to address 0x1xxxx.
  This option is used to determine whether a symbol in an absolute addressing section is in the mirror source area or to notify the linker of the address where the mirror source section is allocated.
  When the CPU core is RL78-S1, the mirror source section is fixed at address 0x0xxxx and this option setting is not necessary. If 1 is specified in this case, an error will occur.
- When common is specified, reference to a symbol allocated to the mirror source area is not supported. Mirror conversion of mirror source addresses is also not supported.
- When the -cpu=S1 option is specified and if 1 is specified in this option, an error will occur.
- When both the -dev option and this option are specified, the specified allocation address is checked against the on-chip ROM (CodeFlash) address range.
- When the -mirror_source=common option is specified and if the -mirror_region option is specified, an error will occur.

[Example of use]

- To specify 0x0xxxx as the address where the mirror source section is allocated, describe as:

```
>asrl -dev=dr5f100pj.dvf -mirror_source=0 main.asm
```

-mirror_region

This option specifies the address range of the mirror destination area.

[Specification format]

```
-mirror_region=start_address,end_address
```

- Interpretation when omitted
  When the -dev option is not specified, a device without a mirror area is assumed.

[Detailed description]

- This option specifies the address range (start address and end address) of the mirror destination area.
- This option is used to calculate the address range of the mirror source area.
  0xF8000 is subtracted from the specified addresses when the CPU core type is RL78-S1, or 0xF0000 is subtracted when the CPU core type is RL78-S2 or RL78-S3. The obtained addresses are assumed as the address range of the mirror source area. If a value outside the range from 0xF0000 to 0xFFFFF is specified, an error will occur.
- If both the -dev option and this option are specified, an error will occur.
- When the -cpu option is specified but this option is not specified, a device without a mirror area is assumed.
- When both the -mirror_source=common option and this option are specified, an error will occur.

[Example of use]

- To specify the address range of the mirror destination area.

```
>asrl -cpu=S2 -mirror_region=0xf3000,0xfaeff main.asm
```

Optimization

The optimization option is as follows.

- -goptimize

-goptimize

This option generates the information for inter-module optimization.

[Specification format]

```
-goptimize
```

- Interpretation when omitted
  The information for inter-module optimization is not generated.

[Detailed description]

- This option generates the additional information for inter-module optimization in the output file.
- At linkage, inter-module optimization is applied to files for which this option has been specified.
  For details on inter-module optimization, see the description of the link option -Optimize.

[Example of use]

- To generate the information for inter-module optimization, describe as:

```
>asrl -goptimize -dev=dr5f100pj.dvf main.asm
```

Symbol definition specification

The symbol definition specification options are as follows.

- -define
- -undefine

-define

This option defines a user-defined assembler symbol (name).

[Specification format]

```
-define=name[=def][,name[=def]]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies *name* as a user-defined assembler symbol (name).
- Specification of *def* is as follows.
  - Only integer values can be specified.
  - If a value other than an integer is specified, 0 is assumed.
  - Integer values can be specified in decimal notation, octal notation with the prefix method (0 ...), and hexadecimal notation (0x ...).
  - Only a negative (-) sign (not positive (+)) can be specified at the beginning of the value.
  - A negative number is converted to a two's complement value.
- This is equivalent to adding "*name* .SET def" at the beginning of the assembly source program.
- An error will occur if *name* is omitted.
- If "=*def*" is omitted, *def* is regarded as 1.
- This option can be specified more than once.
- If both this option and -undefine option are specified, the option specified last will be valid.

[Example of use]

- To define "sample=256" as an assembler symbol, describe as:

```
>asrl -define=sample=256 -dev=dr5f100pj.dvf main.asm
```

-undefine

This option deletes the assembler symbol definition by the -define option.

[Specification format]

```
-undefine=name[,name]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option cancels the definition of user-defined assembler symbol *name* specified by the -define option.
- An error will occur if *name* is omitted.
- This option cannot delete the definition by describing "*name* .EQU *def*".
- This option can be specified more than once.
- If both this option and -define option are specified, the option specified last will be valid.

[Example of use]

- To delete the definition of assembler symbol "test" by the -define option, describe as:

```
>asrl -define=test -dev=dr5f100pj.dvf main.asm -undefine=test
```

Include file reading path specification

The include file reading path specification option is as follows.

- -include

-include

This option specifies the folder to search include files.

[Specification format]

```
-include=path[,path]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies *path* as the folder where the include file is searched for when the include file name is specified without a path or as a relative path in the $INCLUDE or $BINCLUDE control instruction.
  Include files are searched according to the following sequence.
  - <1> Path specified by this option (If multiple paths are specified, they are searched in the order in which they were specified on the command line (that is, from left to right).)
  - <2> Folder that contains the source file where the $INCLUDE or $BINCLUDE control instruction is specified.
  - <3> Current folder (asrl startup folder)
- An error will occur if *path* is omitted.

[Example of use]

- To search include files from folder "D:\include", "D:\src", and the current folder in that order, describe as:

```
>asrl -include=D:\include -dev=dr5f100pj.dvf D:\src\main.asm
```

Input file control

The input file control options are as follows.

- -character_set
- -base_number

-character_set

This option specifies the Japanese/Chinese character code.

[Specification format]

```
-character_set={none|sjis|euc_jp|utf8|big5|gb2312}
```

- Interpretation when omitted
  Processing of Japanese/Chinese character encoding is not performed.

[Detailed description]

- This option specifies the character code to be used for Japanese/Chinese comments and character strings in the input file.
- The parameters that can be specified are shown below.
  An error will occur if any other item is specified.
  Operation is not guaranteed if the specified character code differs from the character code of the input file.

none	Does not process the Japanese and Chinese character code
euc_jp	EUC (Japanese)
sjis	SJIS
utf8	UTF-8
big5	Traditional Chinese
gb2312	Simplified Chinese

- An error will occur if the parameter is omitted.

[Example of use]

- To specify EUC as the character code to be used for Japanese comments and character strings in the input file, describe as:

```
>asrl -character_set=euc_jp -dev=dr5f100pj.dvf main.asm
```

-base_number

This option specifies the notation of the radix for numeric constants.

[Specification format]

```
-base_number={prefix|suffix}
```

- Interpretation when omitted
  It is the same result as when the -base_number=prefix option is specified.

[Detailed description]

- This option specifies the notation of the radix for numeric constants.
- The parameters that can be specified are shown below.
  An error will occur if any other item is specified.

prefix	Specifies the prefix notation (0xn...n).
suffix	Specifies the suffix notation (n...nH).

- An error will occur if the parameter is omitted.

[Example of use]

- To specify the suffix notation of the radix for numeric constants, describe as:

```
>asrl -base_number=suffix -dev=dr5f100pj.dvf main.asm
```

Assembler transition support

The assembler transition support option is as follows.

- -convert_asm

-convert_asm

This option enables the assembler transition supporting function.

[Specification format]

```
-convert_asm
```

- Interpretation when omitted
  The assembler transition supporting function is not enabled.

[Detailed description]

- For the descriptions on "CA78K0R Assembler Language Specifications" in the table below, the descriptions on "RL78 Assembler Language Specifications" should be read instead.

Table 2.7 Assembler transition supporting function

Classification	CA78K0R Assembler Language Specifications	RL78 Assembler Language Specifications	Necessity for Modifying the Source and Specifying Options
Numerical constant	n...nB (n = 0, 1) (binary)	Same as left	Specify -base_number=suffix.
	n...nO (n = 0 to 7) (octal)	Same as left	Specify -base_number=suffix.
	n...nH (n = 0 to 9, A to F, a to f) (hexadecimal)	Same as left	Specify -base_number=suffix.
String	'Character ... Character'	"Character ... Character"	Change two consecutive single quotation marks (') in each string to (\'), and enclose the string with double quotation marks("). Example: .DB 'abc"de' -> .DB "abc\'de"
Operand column	Special function register (SFR, 2nd SFR)	Same as left	Specify -dev.

Classification	CA78K0R Assembler Language Specifications	RL78 Assembler Language Specifications	Necessity for Modifying the Source and Specifying Options
Segment definition directive	No segment	.CSEG TEXT	
	No CSEG relocation attribute	.CSEG TEXTF	
	CSEG CALLT0	.CSEG CALLT0	
	CSEG FIXED	.CSEG TEXT	
	CSEG BASE	.CSEG TEXT	
	CSEG AT	.CSEG AT	
	CSEG UNIT	.CSEG TEXTF	
	CSEG UNITP	.CSEG TEXTF	Add .ALIGN 2.
	CSEG IXRAM	.CSEG TEXTF	
	CSEG OPT_BYTE	.CSEG OPT_BYTE	
	CSEG SECUR_ID	.CSEG SECUR_ID	
	CSEG PAGE64KP	.CSEG TEXTF_UNIT64KP	
	CSEG UNIT64KP	.CSEG TEXTF_UNIT64KP	
	CSEG MIRRORP	.CSEG CONST	
	No DSEG relocation attribute	.DSEG BSSF	
	DSEG SADDR	.DSEG SBSS	
	DSEG SADDRP	.DSEG SBSS	
	DSEG AT	.DSEG BSS_AT	
	DSEG UNIT	.DSEG BSS	
	DSEG UNITP	.DSEG BSS	
	DSEG IHRAM	.DSEG BSS	
	DSEG LRAM	.DSEG BSS	
	DSEG DSPRAM	.DSEG BSS	
	DSEG IXRAM	.DSEG BSS	
	DSEG BASEP	.DSEG BSS	
	DSEG PAGE64KP	.DSEG BSS	
	DSEG UNIT64KP	.DSEG BSS	
	No BSEG relocation attribute	.BSEG SBSS_BIT	
	BSEG UNIT	.BSEG SBSS_BIT	
	BSEG AT	.BSEG BIT_AT	
	ORG	.ORG	

Classification	CA78K0R Assembler Language Specifications	RL78 Assembler Language Specifications	Necessity for Modifying the Source and Specifying Options
Symbol definition directive	EQU	.EQU	A relocatable label cannot be written for an operand.
	SET	.SET	
Memory initialization and area allocation directive	DB	.DB	The code should be changed for size specifications.
	DW	.DB2	The code should be changed for size specifications. If the operand is a string constant, change it to a string. Example: DW 'ab' -> .DB "ba"
	DG	.DB4	The code should be changed for size specifications. If the operand is a string constant, change it to a string. Example: DG 'ab' -> .DB "ba\0\0"
	DS	.DS	
	DBIT	.DBIT	
Linkage directive	PUBLIC	.PUBLIC	
	EXTRN	.EXTERN	
	EXTBIT	.EXTBIT	
Object module name declaration directive	NAME	Commented out	
Branch instruction automatic selection directive	BR	BR !!addr20	
	CALL	CALL !!addr20	
Assemble end directive	END	Commented out	Invalidate this because the code after END becomes valid.
Assemble product type specification control instruction	$PROCESSOR($PC)	Commented out	Specify -dev.
Debug information output control instruction	$DEBUG($DG)	Commented out	Specify -debug.
	$NODEBUG($NODG)	Commented out	Specify -debug.
	$DEBUGA	Commented out	Specify -debug.
	$NODEBUGA	Commented out	Specify -debug.
Cross reference list output specification control instruction	$XREF($XR)	Commented out	
	$NOXREF($NOXR)	Commented out	
	$SYMLIST	Commented out	
	$NOSYMLIST	Commented out	
Include control instruction	$INCLUDE($IC)	$INCLUDE	

Classification	CA78K0R Assembler Language Specifications	RL78 Assembler Language Specifications	Necessity for Modifying the Source and Specifying Options
Assemble list control instruction	$EJECT($EJ)	Commented out	
	$LIST($LI)	Commented out	
	$NOLIST($NOLI)	Commented out	
	$GEN	Commented out	
	$NOGEN	Commented out	
	$COND	Commented out	
	$NOCOND	Commented out	
	$TITLE($TT)	Commented out	
	$SUBTITLE($ST)	Commented out	
	$FORMFEED	Commented out	
	$NOFORMFEED	Commented out	
	$WIDTH	Commented out	
	$LENGTH	Commented out	
	$TAB	Commented out	
Conditional assemble control instruction	$IF(switch name)	Same as left	Specify -define=switch name=1 or -define=switch name=0.
	$IF(switch name : switch name ...)	$IF(switch name | switch name ...)	Specify -define=switch name=1 or -define=switch name=0. Another method is to add "switch name .SET 1" or "switch name .SET 0".
	$_IF	$IF	
	$ELSEIF(switch name : switch name ...)	$ELSEIF(switch name | switch name ...)	Specify -define=switch name=1 or -define=switch name=0. Another method is to add "switch name .SET 1" or "switch name .SET 0".
	$_ELSEIF	$ELSEIF	
	$SET	Commented out	
	$RESET	Commented out	
Kanji code control instruction	$KANJICODE	Commented out	Specify -character_set.
RAM area allocation specification control instruction	$RAM_ALLOCATE	Commented out	Allocate the target segment using ".CSEG TEXTF_UNIT64KP".
Other control instructions	$TOL_INF	Commented out	
	$DGS	Commented out	
	$DGL	Commented out	

[Caution]

- The language specifications of the CA78K0R assembler which are not listed in the above table require the source program to be modified.
- A relocatable label cannot be written for an operand of the symbol definition directive .EQU.
  In this case, replace the reference to the name in the left side of EQU with the relocatable label, and delete the .EQU directive.

  Example 1.

```
DMAINP  DSEG   SADDRP
RABUF1: DS     8
RABUF2: DS     8
OFFSET  EQU    RABUF2 - RABUF1  ; E0551203: Relocatable symbol is not allowed.
FPREAD  EQU    RABUF1.4         ; E0551203: Relocatable symbol is not allowed.
        CSEG
        ADD    A,#OFFSET
        CLR1   FPREAD
        END
```

  (Modification method)

```
OFFSET  EQU    RABUF2 - RABUF1
FPREAD  EQU    RABUF1.4
```

  Disable the above code.

```
        ADD    A,#OFFSET
        CLR1   FPREAD
```

  Modify the above code to the code below.

```
        ADD    A,#RABUF2 - RABUF1
        CLR1   RABUF1.4
```

- When the address width exceeds 16 bits, error occurs at linkage.
  In this case, add a LOWW operator to an address.

  Example 2.

```
DMAINP  DSEG   SADDRP
RABUF1: DS     8
        CSEG
        MOVW   HL,#RABUF1  ; E0562330:Relocation size overflow
        END
```

  (Modification method)

```
        MOVW   HL,#RABUF1
```

  Modify the above code to the code below.

```
        MOVW   HL,#LOWW RABUF1
```

- Operand "(size)" of memory initialization and area allocation directive is different.
  Correct according to the following Example 3.

  Example 3.

```
        CSEG
        DW     (3)
        END
```

(Modification method)

```
        .CSEG
        .DS     6
        .END
```

CA78K0R Assembler: initialize 6 bytes (3 words) area to 00H.
RL78 Assembler: initialize 2 bytes (1 word) area to 03H.

- Change operation not allowed by RL78 Assembler to other method.

Example 4.

```
MSGDATA CSEG   AT 80H
TMSGOK:
        DB     'OK'
        CSEG
        MOV    H,#LOWW TMSGOK/100H   ; E0551215: Illegal label reference.
                                     ; E0550250: Illegal syntax (100H).
        END
```

(Modification method)

```
        MOV    H,#LOWW TMSGOK/100H
```

Modify the above code to the code below.

```
        MOV    H,#HIGH TMSGOK
```

- Change Section definition directives "CSEG UNITP" to ".CSEG TEXTF" + ".ALIGN 2".

Example 5.

```
XMAIN2  CSEG   UNITP
TINTVL: DW     47999
        END
```

(Modification method)

```
XMAIN2  CSEG   UNITP
```

Modify the above code to the code below.

```
XMAIN2  .CSEG   TEXTF
        .ALIGN  2
```

[Example of use]

- To enable the function for supporting transition of programs written for ca78k0r, describe as:

```
>asrl -convert_asm -cpu=S2 -dev=dr5f100pj.dvf main.asm
```

Error message file output specification

The error message file output specification option is as follows.

- -error_file

-error_file

This option outputs error messages to a file.

[Specification format]

```
-error_file=file
```

- Interpretation when omitted
  Error messages are output to only the standard error output.

[Detailed description]

- This option outputs error messages to the standard error output and file *file*.
- If *file* already exists, it will be overwritten.
- An error will occur if *file* is omitted.

[Example of use]

- To output error messages to the standard error output and file "err", describe as:

```
>asrl -error_file=err -dev=dr5f100pj.dvf main.asm
```

Warning message output control

The warning message output control options are as follows.

- -warning
- -no_warning

-warning

This option outputs the specified warning message.

[Specification format]

```
-warning={num|num1-num2}[, ...]
```

- Interpretation when omitted
  All warning messages are output.

[Detailed description]

- This option outputs the specified warning message.
- Specify the error numbers as *num*, *num1*, and *num2*.
  If the error number that does not exist, it will be ignored.
- An error will occur if *num*, *num1*, or *num2* is omitted.
- If *num1-num2* is specified, it is assumed that error numbers within the range have been specified.
- The error number specified by this option is the rightmost 5 digits of the 7-digit number following the "W".
- This option can only control output for warning messages with message numbers (here written with the component number) in the range from 0550000 to 0559999.

[Example of use]

- To output warning message "W0550001" and "W0550005", describe as:

```
>asrl -waning=50001,50005 -dev=dr5f100pj.dvf main.asm
```

-no_warning

This option suppresses outputting warning messages of the specified number.

[Specification format]

```
-no_warning={num|num1-num2}[, ...]
```

- Interpretation when omitted
  All warning messages are output.

[Detailed description]

- This option suppresses outputting warning messages of the specified number.
- Specify the error numbers as *num*, *num1*, and *num2*.
  If the error number that does not exist, it will be ignored.
- An error will occur if *num*, *num1*, or *num2* is omitted.
- If *num1-num2* is specified, it is assumed that error numbers within the range have been specified.
- The error number specified by this option is the rightmost 5 digits of the 7-digit number following the "W".
- This option can only control output for warning messages with message numbers (here written with the component number) in the range from 0550000 to 0559999.

[Example of use]

- To suppress outputting warning message "W0550001" and "W0550005", describe as:

```
>asrl -no_waning=50001,50005 -dev=dr5f100pj.dvf main.asm
```

Subcommand file specification

The subcommand file specification option is as follows.

- @

@

This option specifies a subcommand file.

[Specification format]

```
@file
```

- Interpretation when omitted
  Only the options and file names specified on the command line are recognized.

[Detailed description]

- This option handles *file* as a subcommand file.
- An error will occur if *file* does not exist.
- An error will occur if *file* is omitted.
- See "2.4.2 Subcommand file usage" for details about a subcommand file.

[Example of use]

- To handle "command.txt" as a subcommand file, describe as:

```
>asrl @command.txt main.asm
```

2.5.3 Link options

This section explains options for the link phase.

Caution about options are shown below.

- Uppercase characters and lowercase characters are not distinguished for options.
- Uppercase characters in options and parameters indicate that they can be specified as abbreviations for options and parameters.
  The characters after the uppercase characters can be omitted.

  Example For example, -FOrm=Absolute can be specified as follows.
  -fo=a
  -fo=abs
  -for=absolu

- When a file name is specified as a parameter, "(" and ")" cannot be used.
- When link options are specified for the ccrl command, the -lnkopt option must be used.

The types and explanations for options are shown below.

Table 2.8 Link Options

Classification	Option	Description
Input control	-Input	This option specifies the input file.
	-LIBrary	This option specifies the input library file.
	-Binary	This option specifies the input binary file.
	-DEFine	This option defines an undefined symbol forcedly.
	-ENTry	This option specifies the execution start address.
	-ALLOW_DUPLICATE_MODULE_NAME [V1.09 or later]	This option allows multiple same module names to be specified.

Classification	Option	Description
Output control	-FOrm	This option specifies the output format.
	-DEBug	This option outputs debug information to the output file.
	-NODEBug	This option does not output the debug information.
	-RECord	This option specifies the size of the data record to be output.
	-END_RECORD [V1.05 or later]	This option specifies the end record.
	-ROm	This option specifies the section that maps symbols from ROM to RAM.
	-OUtput	This option specifies the output file.
	-SPace	This option fills the vacant area of the output range.
	-Message	This option output information messages.
	-NOMessage	This option suppresses the output of information messages.
	-MSg_unused	This option notifies the symbol that is not referenced.
	-BYte_count	This option specifies the maximum byte count for a data record.
	-FIX_RECORD_LENGTH_AND_ALIGN [V1.06 or later]	Fixes the format of data records to be output.
	-PADDING	This option fills in data at the end of a section.
	-CRc	This option specifies whether to perform the CRC operation.
	-VECT	This option stores an address value in the unused areas in the vector table.
	-VECTN	This option stores address values in the specified areas in the vector table.
	-SPLIT_VECT [V1.07 or later]	This option generates split vector table sections.
	-VFINFO	This option outputs the variable/function information file.
	-CFI [Professional Edition only] [V1.06 or later]	Generates the function list for use in detecting illegal indirect function calls.
	-CFI_ADD_Func [Professional Edition only] [V1.06 or later]	Specifies the symbol or address of a function to be added to the function list for use in detecting illegal indirect function calls.
	-CFI_IGNORE_Module [Professional Edition only] [V1.06 or later]	Specifies modules which are to be exempted from the function list for use in detecting illegal indirect function calls.
	-RAM_INIT_TABLE_SECTION [V1.12 or later]	This option generates an information table for RAM initialization.
List output	-LISt	This option outputs the list file.
	-SHow	This option specifies information that is output to the list file.

Classification	Option	Description
Optimization	-Optimize	This option specifies whether to execute inter-module optimization.
	-NOOptimize	This option disables inter-module optimization.
	-SEction_forbid	This option disables optimization for the specified section.
	-Absolute_forbid	This option disables optimization regarding address + size specification.
	-SYmbol_forbid [V1.02 or later]	This option specifies unreferenced symbols that are not to be deleted.
	-ALLOW_OPTIMIZE_ENTRY_BLOCK [V1.13 or later]	This option performs optimization on the areas that are allocated before the execution start symbol.
Section specification	-STARt	This option specifies the start address of the section.
	-FSymbol	This option outputs external defined symbols to the symbol address file.
	-USER_OPT_BYTE	This option specifies the value set for the user option bytes.
	-OCDBG	This option specifies the control value for the on-chip debug.
	-SECURITY_OPT_BYTE [V1.12 or later]	This option specifies the control value for the security option byte.
	-SECURITY_ID	This option specifies a security ID value.
	-FLASH_SECURITY_ID [V1.12 or later]	This option specifies the value to be set for the flash programmer security ID.
	-AUTO_SECTION_LAYOUT	This option automatically allocates sections.
	-SPLIT_SECTION [V1.12 or later]	This option enables automatic allocation of sections for each module.
	-STRIDE_DSP_MEMORY_AREA [V1.12 or later]	This option enables automatic allocation of sections to areas split by the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).
	-DEBUG_MONITOR	This option specifies the OCD monitor area.
	-RRM	This option specifies the work area for the RRM/DMM function.
	-SELF	This option disables allocation of a section to the self RAM area.
	-SELFW	This option outputs a warning message when a section is allocated to the self RAM area.
	-OCDTR	This option disables allocation of a section to the trace RAM and self RAM areas.
	-OCDTRW	This option outputs a warning message when a section is allocated to the trace RAM and self RAM areas.

Classification	Option	Description
Section specification	-OCDHPI	This option disables allocation of a section to the hot plug-in RAM, trace RAM, and self RAM areas.
	-OCDHPIW	This option outputs a warning message when a section is allocated to the hot plug-in RAM, trace RAM, and self RAM areas.
	-DSP_MEMORY_AREA [V1.12 or later]	This option disables allocation of a section to the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).
Verify specification	-CPu	This option checks the consistency of the address to which the section is allocated.
	-CHECK_DEVICE	This option checks the device file specified when creating an object file.
	-CHECK_64K_ONLY	This option disables checking whether an allocated section exceeds the (64K-1)-byte boundary.
	-NO_CHECK_SECTION_LAYOUT	This option disables checking of the consistency between the address to which the section is allocated and the address information in a device file.
	-CHECK_OUTPUT_ROM_AREA [V1.07 or later]	This option checks whether the output address of a HEX file ranges in internal ROM or the data flash area.
Subcommand file specification	-SUbcommand	This option specifies options with a subcommand file.
Microcontroller specification	-DEVICE	This option specifies the device file name.

Classification	Option	Description
Other	-S9	This option outputs the S9 record at the end.
	-STACk	This option outputs the stack information file.
	-COmpress	This option compresses the debug information.
	-NOCOmpress	This option does not compress the debug information.
	-MEMory	This option specifies the memory size occupied during linking.
	-REName	This option changes an external symbol name or a section name.
	-LIB_REName [V1.08 or later]	This option changes the name of a symbol or section that was input from a library.
	-DELete	This option deletes an external symbol name or a library module.
	-REPlace	This option replaces library modules.
	-EXTract	This option extracts library modules.
	-STRip	This option deletes debug information in the load module file or library file.
	-CHange_message	This option changes the type of information, warning, and error messages.
	-Hide	This option deletes local symbol name information from the output file.
	-Total_size	This option displays the total size of sections after the linking to the standard error output.
	-VERBOSE [V1.10 or later]	This option displays detailed information in the standard error output.
	-LOgo	This option outputs the copyright notice.
	-NOLOgo	This option suppresses the output of the copyright notice.
	-END	This option executes option strings specified before this option.
	-EXIt	This option specifies the end of option specifications.

Input control

The input control options are as follows.

- -Input
- -LIBrary
- -Binary
- -DEFine
- -ENTry
- -ALLOW_DUPLICATE_MODULE_NAME [V1.09 or later]

-Input

This option specifies the input file.

[Specification format]

```
-Input=suboption[{,|∆} ...]
  suboption := {file|file(module[, ...])}
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies input file *file*.
  If multiple files are specified, delimit them with a comma (,) or space.
- Wildcard characters (*, ?) can also be used.
  The character strings specified with wildcard characters are expanded in alphabetical order.
  Expansion of numerical values precedes that of alphabetic characters. Uppercase characters are expanded before lowercase characters.
- Files that can be specified as input files are object files output from the compiler or the assembler and relocatable files, load module files, Intel HEX files, and Motorola S-record files output from the optimizing linker.
  In addition, a module in a library can be specified using the format of "*library(module*)".
  Specify the module name without the extension.
- If no extension is specified for the input filename, then if no module name is specified, it is assumed to be ".obj"; if a module name is specified, it is assumed to be ".lib".

[Caution]

- This option can be used only in a subcommand file.
  An error will occur if this option is specified on the command line.
  When input files are specified on the command line, specify them without the -input option.

[Example of use]

- To input a.obj and module "e" in lib1.lib, describe as:
  <Command line>

```
>rlink -subcommand=sub.txt
```

<Subcommand file "sub.txt">

```
-input=a.obj lib1(e)
```

- To input all ".obj" files beginning with "c", describe as:
  <Command line>

```
>rlink -subcommand=sub.txt
```

<Subcommand file "sub.txt">

```
-input=c*.obj
```

[Remark]

- If the -form=object or -extract option is specified, this option will be invalid.
- If an Intel HEX file is specified as an input file, only the -form=hexadecimal option can be specified. If a Motorola S-record file is specified, only the -form=stype option can be specified.
  If the output file name is not specified, it will be "*first input file name_combine.extension*" (If the input file is "a.mot", the output file will be "a_combine.mot").

-LIBrary

This option specifies the input library file.

[Specification format]

```
-LIBrary=file[,file]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies input library file *file*.
  If multiple files are specified, delimit them with a comma (,).
- Wildcard characters (*, ?) can also be used.
  The character strings specified with wildcard characters are expanded in alphabetical order.
  Expansion of numerical values precedes that of alphabetic characters. Uppercase characters are expanded before lowercase characters.
- If the extension is omitted from the input file specification, it is assumed that ".lib" has been specified.
- If this option and the -form=library or -extract option are specified at the same time, the specified library file is input as the target library to be edited.
  Otherwise, undefined symbols are searched in the library file after the link processing between files specified as the input files are executed.
- The symbols are searched in the library file in the following sequence:
  - User library files specified by this option (in the specified order)
  - System library files specified by this option (in the specified order)
  - Default library (environment variables "HLNK_LIBRARY1", "HLNK_LIBRARY2", and "HLNK_LIBRARY3"[Note] in that order)

Note See "2.3 Environment Variable" for details about environment variables.

[Example of use]

- To input a.lib and b.lib, describe as:

```
>rlink main.obj -library=a.lib,b
```

- To input all ".lib" files beginning with "c", describe as:

```
>rlink main.obj -library=c*.lib
```

-Binary

This option specifies the input binary file.

[Specification format]

```
-Binary=suboption[, ...]
  suboption := file(section[:alignment][/attribute][,symbol])
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies input binary file *file*.
  If multiple files are specified, delimit them with a comma (,).
- If the extension is omitted from the input file specification, it is assumed that ".bin" has been specified.
- Input binary data is allocated as the data of specified section *section*.
  Specify the section address by the -start option.
  An error will occur if *section* is omitted.
- When symbol *symbol* is specified, it can be linked as a defined symbol.
  For a variable name referenced by a C program, add "_" at the head of the reference name in the program.
- The section specified by this option can have its section attribute and number of alignment specified.
- CODE or DATA can be specified as section attribute *attribute*.
  If *attribute* is omitted, the write, read, and execute attributes will be all valid by default. [V1.04 or earlier]
- CALLT0, CODE, TEXT, TEXTF, TEXTF_UNIT64KP, CONST, CONSTF, SDATA, DATA, DATAF, OPT_BYTE, or SECUR_ID can be specified as the section attribute *attribute*. CODE becomes the same as the relocation attribute of TEXT. If a name other than ".option_byte" is specified for the section name while OPT_BYTE is specified, an error will occur. If a name other than ".security_id" is specified for the section name while SECUR_ID is specified, an error will occur.
  If *attribute* is omitted, the write, read, and execute attributes will all be valid by default. [V1.05 or later]
- The value that can be specified for number of alignment *alignment* is a power of 2 (1, 2, 4, 8, 16, or 32).
  Other value cannot be specified.
  If *alignment* is omitted, "1" will be valid by default.

[Restrictions]

- The binary file specified by this option can be allocated to only addresses 0 to 0x0FFFF. [V1.04 or earlier]
  Generate an assembler source code like that shown below to change the section attribute to the desired attribute, such as for allocating the binary file to an address greater than address 0x10000.

```
    .SECTION BIN_SEC, TEXTF
    $BINCLUDE(tp.bin)
```

[Example of use]

- b.bin is allocated from 0x200 as the .D1bin section.
  c.bin is allocated after .D1bin as the .D2bin section (with the number of alignment = 4).
  The c.bin data is linked as defined symbol "_datab".
  To perform the above operations, describe as:

```
>rlink a.obj -start=.D*/200 -binary=b.bin(.D1bin),c.bin(.D2bin:4,_datab)
```

[Remark]

- If the -form={object|library} option or -strip option is specified, this option will be invalid.
- If input object file is not specified, this option cannot be specified.

-DEFine

This option defines an undefined symbol forcedly.

[Specification format]

```
-DEFine=suboption[, ...]
  suboption := {symbol1=symbol2|symbol1=value}
```

- Interpretation when omitted
  None

[Detailed description]

- This option defines undefined symbol *symbol1* forcedly as external defined symbol *symbol2* or numerical value *value*.
- Specify *value* in hexadecimal.
  If the specified value starts with a character from A to F, symbols are searched first, and if corresponding symbol is not found, the value is interpreted as a numerical value.
  Values starting with 0 are always interpreted as numerical values.
- If the specified symbol name is a C variable name, add "_" at the head of the definition name in the program.

[Example of use]

- To define "_sym1" as the same value as external defined symbol "_data", describe as:

```
>rlink -define=_sym1=_data a.obj b.obj
```

- To define "_sym2" as 0x4000, describe as:

```
>rlink -define=_sym2=4000 a.obj b.obj
```

[Remark]

- If the -form={object|relocate|library} option is specified, this option will be invalid.

-ENTry

This option specifies the execution start address.

[Specification format]

```
-ENTry={symbol|address}
```

- Interpretation when omitted
  None

[Detailed description]

- This option defines execution start address with external defined symbol *symbol* or address *address*.
- Specify *address* in hexadecimal.
  If the specified value starts with a character from A to F, defined symbols are searched first, and if corresponding symbol is not found, the value is interpreted as an address.
  Values starting with 0 are always interpreted as addresses.
- If the specified symbol name is a C variable name, add "_" at the head of the definition name in the program.
- This option setting takes priority over the entry symbol specified at compilation.

[Example of use]

- To specify main function in C as the execution start address, describe as:

```
>rlink -entry=_main a.obj b.obj
```

- To specify 0x100 as the execution start address, describe as:

```
>rlink -entry=100 a.obj b.obj
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- Be sure to specify *symbol* if you intend to enable inter-module optimization (-optimize[={symbol_delete|speed}]). If *address* is specified with this option, optimization regarding deletion of unreferenced symbols will be disabled.
- If the address specified by the -entry option is included in any of the sections allocated by the -start option, optimization in the range from the first address of the section up to the address specified by the -entry option will be suppressed.

-ALLOW_DUPLICATE_MODULE_NAME [V1.09 or later]

This option allows a library to be generated from multiple same module names.

[Specification format]

```
-ALLOW_DUPLICATE_MODULE_NAME
```

- Interpretation when omitted
  None

[Detailed description]

- This option allows multiple input files with the same module name to be specified to generate a library.
- If the library already contains a module having the same name with other modules to be registered in the library, the other modules are renamed by adding a postfix number ".<N>".
- <N> is assigned a number as a unique module name in the generating library. If can't assigned a unique number, The linker will output the error message and quit.

[Example of use]

- To generate a library a.lib from multiple input files having the same module name (mod), describe as:

```
> rlink -allow_duplicate_module_name -form=lib -output=a.lib b\mod.obj c\mod.obj
d\mod.obj
```

The command line above leads to generate a library a.lib containing the following modules:

- mod (originally b\mod.obj)
- mod.1 (originally c\mod.obj)
- mod.2 (originally d\mod.obj)

[Remark]

- If the -form={ object|absolute|relocate|hexadecimal|stype|binary }, -strip, or -extract option is specified, this option will be invalid.

Output control

The output control options are as follows.

- -FOrm
- -DEBug
- -NODEBug
- -RECord
- -END_RECORD [V1.05 or later]
- -ROm
- -OUtput
- -SPace
- -Message
- -NOMessage
- -MSg_unused
- -BYte_count
- -FIX_RECORD_LENGTH_AND_ALIGN [V1.06 or later]
- -PADDING
- -CRc
- -VECT
- -VECTN
- -SPLIT_VECT [V1.07 or later]
- -VFINFO
- -CFI [Professional Edition only] [V1.06 or later]
- -CFI_ADD_Func [Professional Edition only] [V1.06 or later]
- -CFI_IGNORE_Module [Professional Edition only] [V1.06 or later]
- -RAM_INIT_TABLE_SECTION [V1.12 or later]

-FOrm

This option specifies the output format.

[Specification format]

```
-FOrm=format
```

- Interpretation when omitted
  A load module file is output (It is the same result as when the -form=absolute option is specified).

[Detailed description]

- This option specifies output format *format*.
- The items that can be specified as *format* are shown below.

Absolute	Outputs a load module file.
Relocate	Outputs a relocatable file.
Object	Outputs an object file. Use this when a module is extracted as an object file from a library by the -extract option.
Library[={S|U}]	Outputs a library file. When "library=s" is specified, a system library file is output. When "library=u" is specified, a user library file is output. If only "library" is specified, it is assumed that "library=u" has been specified.
Hexadecimal	Outputs an Intel HEX file. See "3.5 Intel HEX File" for details.
Stype	Outputs a Motorola S-record file. See "3.6 Motorola S-record File" for details.
Binary	Outputs a binary file.

[Remark]

- The relations between output formats and input files or other options are shown below.

Table 2.9 Relations Between Output Formats And Input Files Or Other Options

Output Format	Specified Option	File Format That Can Be Input	Specifiable Option Note 1
Absolute	-strip specified	Load module file	-input, -output
	Other than above	Object file Relocatable file Binary file Library file	-input, -library, -binary, -debug, -nodebug, -cpu, -start, -rom, -entry, -output, -hide, -optimize, -nooptimize, -symbol_forbid, -section_forbid, -absolute_forbid, -compress, -nocompress, -rename, -lib_rename, -delete, -define, -fsymbol, -stack, -memory, -msg_unused, -show={symbol|reference|xreference|total_size|vector|struct|relocation_attribute|cfi|all}, -user_opt_byte, -ocdbg, -security_id, -device, -padding, -vect, -vectn, -split_vect, -vfinfo, -auto_section_layout, -debug_monitor, -rrm, -self, -selfw, -ocdtr, -ocdtrw, -ocdhpi, -ocdhpiw, -check_device, -check_64k_only, -no_check_section_layout, -cfi, -cfi_add_func, -cfi_ignore_module
Relocate	-extract specified	Library file	-library, -output
	Other than above	Object file Relocatable file Binary file Library file	-input, -library, -debug, -nodebug, -output, -hide, -rename, -lib_rename, -delete, -show={symbol|xreference|total_size|all}, -device, -check_device
Object	-extract specified	Library file	-library, -output
Hexadecimal Stype Binary		Object file Relocatable file Binary file Library file	-input, -library, -binary, -cpu, -start, -rom, -entry, -output, -space, -optimize, -nooptimize, -symbol_forbid, -section_forbid, -absolute_forbid, -rename, -lib_rename, -delete, -define, -fsymbol, -stack, -record, -end_record Note 2, -s9 Note 2, -byte_count Note 3, -fix_record_length_and_align Note 6, -memory, -msg_unused, -show={symbol|reference|xreference|total_size|vector|struct|relocation_attribute|cfi|all}, -user_opt_byte, -ocdbg, -security_id, -crc, -device, -padding, -vect, -vectn, -split_vect, -vfinfo, -auto_section_layout, -debug_monitor, -rrm, -self, -selfw, -ocdtr, -ocdtrw, -ocdhpi, -ocdhpiw, -check_device, -check_64k_only, -no_check_section_layout, -check_output_rom_area Note 7, -cfi, -cfi_add_func, -cfi_ignore_module
		Load module file	-input, -output, -record, -end_record Note 2, -s9 Note 2, -byte_count Note 3, -fix_record_length_and_align Note 6, -show={symbol|reference|xreference|all} -device, -check_output_rom_area Note 7
		Intel HEX file Note 4	-input, -output -device, -check_output_rom_area Note 7
		Motorola S-record file Note 4	-input, -output, -s9 Note 2 -device, -check_output_rom_area Note 7
Library	-strip specified	Library file	-library, -output, -memory Note 5
	-extract specified	Library file	-library, -output
	Other than above	Object file Relocatable file	-input, -library, -output, -hide, -rename, -delete, -replace, -memory Note 5, -show={symbol|section|all}, -allow_duplicate_module_name

Note 1. The following options can always be specified.
-message, -nomessage, -change_message, -logo, -nologo, -form, -list, -subcommand

Note 2. The -end_record and -s9 option are valid only when the -form=stype option is specified.

Note 3. The -byte_count option is valid only when the -form=hexadecimal or -form=stype option is specified.

Note 4. If an Intel HEX file is specified as an input file, only the -form=hexadecimal option can be specified. If a Motorola S-record file is specified, only the -form=stype option can be specified.

Note 5. The -memory option cannot be specified when the -hide option is specified.

Note 6. The -fix_record_length_and_align option is valid only when the -form=hexadecimal or -form=stype option is specified.

Note 7. The -check_output_rom_area option is valid only when the -form=hexadecimal or -form=stype option is specified.

[Example of use]

- To output relocatable file c.rel from a.obj and b.obj, describe as:

```
>rlink a.obj b.obj -form=relocate -output=c.rel
```

- To extract module "a" from lib.lib and output as an object file, describe as:

```
>rlink -library=lib.lib -extract=a -form=object
```

- To extract module "a" from lib.lib and output library file exta.lib, describe as:

```
>rlink -library=lib.lib -extract=a -form=library -output=exta
```

- To extract module "a" from lib.lib and output relocatable file a.rel, describe as:

```
>rlink -library=lib.lib -extract=a -form=relocate
```

-DEBug

This option outputs debug information to the output file.

[Specification format]

```
-DEBug
```

- Interpretation when omitted
  The debug information is output to the output file (It is the same result as when the -debug option is specified).

[Detailed description]

- This option outputs debug information to the output file.

[Example of use]

- To output debug information to the output file, describe as:

```
>rlink a.obj b.obj -debug -output=c.abs
```

[Remark]

- If the -form={object|library|hexadecimal|stype|binary}, -strip option or -extract option is specified, this option will be invalid.
- If two or more output file names are specified using the -form=absolute option and -output option, the debug information will not be output.

-NODEBug

This option does not output the debug information.

[Specification format]

```
-NODEBug
```

- Interpretation when omitted
  The debug information is output to the output file (It is the same result as when the -debug option is specified).

[Detailed description]

- This option does not output the debug information.

[Example of use]

- Not to output the debug information, describe as:

```
>rlink a.obj b.obj -nodebug -output=c.abs
```

[Remark]

- If the -form={object|library|hexadecimal|stype|binary}, -strip option or -extract option is specified, this option will be invalid.

-RECord

This option specifies the size of the data record to be output.

[Specification format]

```
-RECord=record
```

- Interpretation when omitted
  Various data records are output according to each address.

[Detailed description]

- This option outputs data with data record *record* regardless of the address range.
- The items that can be specified as *record* are shown below.

H16	HEX record
H20	Expanded HEX record
H32	32-bit HEX record
S1	S1 record
S2	S2 record
S3	S3 record

- If there is an address that is larger than the specified data record, the appropriate data record is selected for the address.

[Example of use]

- To output 32-bit HEX record regardless of the address range:

```
>rlink a.obj b.obj -record=H32 -form=hexadecimal -output=c.hex
```

[Remark]

- If the -form={hexadecimal|stype} option is not specified, this option will be invalid.
- An error will occur if the -record={S1|S2|S3} option is specified when the -form=hexadecimal option is specified, or if the -record={H16|H20|H32} option is specified when the -form=stype option is specified.

-END_RECORD [V1.05 or later]

This option specifies the end record.

[Specification format]

```
-END_RECORD=record
```

- Interpretation when omitted
  The end record is output to suit the address of the entry point.

[Detailed description]

- This option specifies the type of end record for a Motorola S-record file.
- The following can be specified for *record*.

S7	S7 record
S8	S8 record
S9	S9 record

- When the entry point address is larger than the specified address field, select an end record to suit the address of the entry point.

[Example of use]

- To output a 32-bit S-type end record regardless of the address range, write this as:

```
> rlink a.obj b.obj -end_record=S7 -form=stype -output=c.mot
```

[Remark]

- When -form={stype} is not specified, this option outputs an error message and terminates execution.

-ROm

This option specifies the section that maps symbols from ROM to RAM.

[Specification format]

```
-ROm=ROMsection=RAMsection[,ROMsection=RAMsection]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option reserves ROM and RAM areas in the initialized data area and relocates defined symbols in the ROM section with the address in the RAM section.
- Specify a relocatable section including the initial value for ROM section *ROMsection*.
- Specify a nonexistent section or relocatable section whose size is 0 for RAM section *RAMsection*.
- A wildcard symbol (*) can be used in *ROMsection* and *RAMsection*. [V1.13 or later]
  - If the name of a relocatable ROM section with the initial value matches the wildcard expression of *ROMsection*, the name is processed as a RAM section name. At this time, a wildcard symbol (*) in *RAMsection* is replaced with the part that matches the wildcard symbol (*) in the ROM section name.

    Example When there are four ROM sections (.data, .data_1, .sdata, and .sdata_1) and -rom=*data*=*data*_R is specified, four RAM sections (.data_R, .data_1_R, .sdata_R, and .sdata_1_R) are generated.

    Note The RAM section names after replacement must be handled appropriately by using, for example, the -start option.
  - Multiple wildcard symbols (*) can be specified. The number of wildcard symbols must match between *ROMsection* and *RAMsection*.

    Example

```
-rom=.data*=.data*_R      # No problem
-rom=.data*=.data*_R_*    # Error due to too many wildcard symbols in
                          RAMsection
```

  - If a section having the same name as the one generated by replacement already exists, an error occurs.

[Example of use]

- To reserve the .data.R section with the same size as the .data section and relocate defined symbols in the .data section with address in the .data.R section, describe as:

```
>rlink a.obj b.obj -rom=.data=.data.R -start=.data/100,.data.R/FAF00
```

[Remark]

- If the -form={object|relocate|library} option, -strip option, or -extract option is specified, this option will be invalid.

-OUtput

This option specifies the output file.

[Specification format]

```
-OUtput=suboption[, ...]
[V1.06 or earlier]
  suboption := {file|file=range}
  range := {address1-address2|section[: ...]}
[V1.07 or later]
  suboption := {file|file=range|file=range/load-address|file=/load-address}
  range := {address1-address2|section[: ...]}
```

- Interpretation when omitted
  The output file name is "*first-input-file-name.default-extension*".
  The default extensions are shown below.

  When the -form=absolute option is specified: abs
  When the -form=relocate option is specified: rel
  When the -form=object option is specified: obj
  When the -form=library option is specified: lib
  When the -form=hexadecimal option is specified: hex
  When the -form=stype option is specified: mot
  When the -form=binary option is specified: bin

[Detailed description]

- This option specifies output file *file*.
- Specify the start address and end address of the output range in hexadecimal as *address1* and *address2*.
  The output range including "-" is always interpreted as addresses.
- Specify the section to be output as *section*.
  If multiple files are specified, delimit them with a colon (:).
- If this option and the -form={absolute|hexadecimal|stype|binary} option are specified at the same time, two or more files can be specified.
- If "*load-address*" is specified, when outputting an Intel HEX file or Motorola S-record file, the first load address in the file is changed to the value specified with "*load-address*".

[Example of use]

- To output the range from 0 to 0xffff to file1.abs and the range from 0x10000 to 0x1ffff to file2.abs, describe as:

```
>rlink a.obj b.obj -output=file1.abs=0-ffff,file2.abs=10000-1ffff
```

- To output the .sec1 and .sec2 sections to file1.abs and the .sec3 section to file2.abs, describe as:

```
>rlink a.obj b.obj -output=file1.abs=.sec1:.sec2,file2.abs=.sec3
```

[Remark]

- If a input file is an Intel Hex file or Motorola S-record file, two or more output files cannot be specified by this option.
  If this option is omitted, the output file name will be "*first input file name*_combine.*extension*" (If the input file is "a.mot", the output file will be "a_combine.mot").
- "*load-address*" can be specified only when the -form={hexadecimal | stype} option is specified.

-SPace

This option fills the vacant area of memory in the output range.

[Specification format]

```
-SPace[=data]
```

- Interpretation when omitted
  None

[Detailed description]

- This option fills the vacant area of the output range with user-specified data *data*.
- The items that can be specified as *data* are shown below.

Numerical Value	Hexadecimal value
Random	Random number

- The way of filling unused areas differs with the output range specification as follows.
  - When the -Output option is used to specify sections as the range for output:
    The specified numerical value is output to unused areas between the specified sections.
  - When the -Output option is used to specify a range of addresses as the range for output:
    The specified numerical value is output to unused areas within the specified address range.
  - When the -FIX_RECORD_LENGTH_AND_ALIGN option is specified:
    The specified numerical value is output to an unused area at the top of a section, which starts at an address that can be divided by the alignment number.
    The specified numerical value is output when the end of a section does not reach the specified record length.
- Output data sizes in units of 1, 2, or 4 bytes are valid. The size is determined by the hexadecimal number specified using this option.
  If a 3-byte value is specified, the upper digit is extended with 0 to handle it as a 4-byte value.
  If an odd number of digits is specified, the upper digit is extended with 0 to handle it as an even number of digits.
- If the size of a vacant area is not a multiple of the size of the output data, the value is output as many times as possible, and then a warning will be output.

[Example of use]

- To fill the vacant memory area with "0xff" within the range from address 0x100 to address 0x2FF, describe as:

```
>rlink a.obj b.obj -form=hexadecimal -output=file1=100-2ff -start=.SEC1/100,.SEC2/
200 -space=ff
```

[Remark]

- If the specification of the data is omitted in this option, vacant areas are not filled with values.
- This option is valid only when the -form={binary|stype|hexadecimal} option is specified.
- If the output range is not specified in the -output option and the -fix_record_length_and_align option is not specified, this option will be invalid.

-Message

This option output messages of information level.

[Specification format]

```
-Message
```

- Interpretation when omitted
  The output of messages of information level is suppressed (It is the same result as when the -nomessage option is specified).

[Detailed description]

- This option output messages of information level.

[Example of use]

- To output messages of information level, describe as:

```
>rlink a.obj b.obj -message
```

-NOMessage

This option suppresses the output of information messages.

[Specification format]

```
-NOMessage[={num|num-num}[, ...]]
```

- Interpretation when omitted
  The output of information messages is suppressed.

[Detailed description]

- This option suppresses the output of information messages.
- If message number *num* is specified, the output of the message with the specified number is suppressed.
  Also, a range of message numbers can be specified using a hyphen (-).
- Specify the 4-digit number that is output after the component number (05) and the phase of occurrence (6) as *num*
  (for example, specify 0004 for message number M0560004).
  0 at the beginning of the 4-digit number can be omitted (for example, specify 4 for message number M0560004).
- If a number of a warning or error type message is specified, the output of the message is suppressed assuming that -change_message option has changed the specified message to the information type.

[Example of use]

- To suppress outputting messages of M0560004, M0560100 to M0560103, and M0560500, describe as:

```
>rlink a.obj b.obj -nomessage=4,100-103,500
```

-MSg_unused

To output externally defined symbols that are not referenced.

[Specification format]

```
-MSg_unused
```

- Interpretation when omitted
  None

[Detailed description]

- This option is used to detect externally defined symbols that have not been referenced even once during the link processing.

[Example of use]

- To output externally defined symbols that are not referenced.

```
>rlink a.obj b.obj -message -msg_unused
```

[Remark]

- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.
- If a load module file is input, this option will be invalid.
- This option must be specified together with the -message option.
- The a message may be output for the function that inline expansion was performed during compilation.
  In this case, add a static declaration for the function definition to suppress the output of the message.
- In either of the following cases, the reference relationship cannot be analyzed correctly and the information notified through an output message will be incorrect.
  - If there are references to constant symbols within the same file
  - When optimization is enabled at compilation and a function directly under another function is called.

-BYte_count

This option specifies the maximum byte count for a data record.

[Specification format]

```
-BYte_count=num
```

- Interpretation when omitted
  When the -form=hexadecimal option is specified, an Intel HEX file is generated assuming that the maximum byte count is "0xFF".
  When the -form=stype option is specified, a Motorola S-record file is generated assuming that the maximum byte count is "0x10".

[Detailed description]

- This option is used to specify the length of data records in Intel HEX files or Motorola S-record files to be generated.
- Values from 01 to FF (hexadecimal) are specifiable for Intel HEX files.
- The following ranges of values are specifiable for Motorola S-record files.
  - S1 records: 01 to FC (hexadecimal)
  - S2 records: 01 to FB (hexadecimal)
  - S3 records: 01 to FA (hexadecimal)

[Example of use]

- To specify 0x10 as the maximum byte count for a data record, describe as:

```
>rlink a.obj b.obj -form=hexadecimal -byte_count=10
```

[Remark]

- When the -form={hexadecimal|stype} option is not specified, this option will be invalid.

-FIX_RECORD_LENGTH_AND_ALIGN [V1.06 or later]

Fixes the format of data records to be output.

[Specification format]

```
-FIX_RECORD_LENGTH_AND_ALIGN=align
```

- Interpretation when omitted
  None

[Detailed description]

- This option is used to output an Intel HEX file or a Motorola S-record file with records of a fixed length starting from the address that has alignment with the specified number.
- The address of the first record to be output should be less than or equal to the first address of a section and be the largest number that can be divided by the specified alignment number.
- The specified numerical value or default value for the parameter of the -BYte_count option will be used as the length of the records.
- Since the length of records is fixed, each record may include data for more than one section.
- In unused areas, the value specified by the -SPace option will be output. If the -SPace option is not specified, 0x00 (with the -CRc option not specified) or 0xff (with the -CRc option specified) as the default value will be output.

[Example of use]

- Starting the output of records from an address that can be divided by 8, with the length of each record fixed to 16 bytes (10 in hexadecimal).

```
>rlink a.obj b.obj -form=hexadecimal -byte_count=10 -fix_record_length_and_align=8
```

[Remark]

- When the -form={hexadecimal|stype} option is not specified, this option will be invalid.

-PADDING

This option fills in data at the end of a section.

[Specification format]

```
-PADDING
```

- Interpretation when omitted
  None

[Detailed description]

- This option fills in data at the end of a section so that the section size is a multiple of the alignment of the section.
- This option fills in padding data only in sections of an instruction, the const variable, and a variable with the initial value. This option does not apply to sections of variables that have no initial values.

[Example of use]

- In the following case, 1 byte of padding data is filled in the .SEC1 section, and linking is performed with a size of 0x06.
  Alignment of the .SEC1 section: 2 bytes
  Size of the .SEC1 section: 0x05 bytes
  Alignment of the .SEC2 section: 1 byte
  Size of the .SEC2 section: 0x03 bytes

```
>rlink a.obj b.obj -start=.SEC1,.SEC2/0 -padding
```

- In the following case, if 1 byte of padding data is filled in the .SEC1 section, and linking is performed with a size of 0x06, then an error will be output because it overlaps with the .SEC2 section.
  Alignment of the .SEC1 section: 2 bytes
  Size of the .SEC1 section: 0x05 bytes
  Alignment of the .SEC2 section: 1 byte
  Size of the .SEC2 section: 0x03 bytes

```
>rlink a.obj b.obj -start=.SEC1/0,.SEC2/5 -padding
```

[Remark]

- The value of the generated padding data is 0x00.
- Since padding is not performed to an absolute address section, the size of an absolute address section should be adjusted by the user.

-CRc

This option specifies whether to perform the CRC operation.

[Specification format]

```
-CRc=address=operation-range[/operation-method][(initial-value)][:endian]
    operation-range:= start-end[,start-end]... [V1.01]
                      {start-end|section}[,{start-end|section}]... [V1.02 or later]
    operation-method:= {16-CCITT-MSB-LITTLE-4|16-CCITT-LSB|SENT-MSB} [V1.01]
                       {CCITT|16-CCITT-MSB|16-CCITT-MSB-LITTLE-4|16-CCITT-MSB-
                       LITTLE-2|16-CCITT-LSB|16|SENT-MSB|32-ETHERNET} [V1.02 or later]
    endian:= {BIG|LITTLE}[-size-offset]
```

- Interpretation when omitted
  The CRC operation and outputting the result are not performed.

[Detailed description]

- CRC (cyclic redundancy check) operation is done for the specified range of load objects in the order from the lower to the higher addresses, and the result is output to the specified output address in the specified endian.
- Specify the output address for *address*.
  The range that can be specified is 0x0 to 0xFFFFF.
- Specify the operation range (start address and end address) for *start* and *end*.
  The range that can be specified is 0x0 to 0xFFFFF.
- Specify one of the following as the operation method.

Operation Method	Description
CCITT [V1.02 or later]	The result of CRC-16-CCITT operation is obtained with the MSB first, an initial value of 0xFFFF, and inverse of XOR performed. The generator polynomial is $x^{16}+x^{12}+x^5+1$.
16-CCITT-MSB [V1.02 or later]	The result of CRC-16-CCITT operation is obtained with the MSB first. The generator polynomial is $x^{16}+x^{12}+x^5+1$.
16-CCITT-MSB-LITTLE-4	The input is handled in little endian in 4-byte units and the result of CRC-16-CCITT operation is obtained with the MSB first. The generator polynomial is $x^{16}+x^{12}+x^5+1$.
16-CCITT-MSB-LITTLE-2 [V1.02 or later]	The input is handled in little endian in 2-byte units and the result of CRC-16-CCITT operation is obtained with the MSB first. The generator polynomial is $x^{16}+x^{12}+x^5+1$.
16-CCITT-LSB	The result of CRC-16-CCITT operation is obtained with the LSB first. The generator polynomial is $x^{16}+x^{12}+x^5+1$.
16 [V1.02 or later]	The result of CRC-16 operation is obtained with the LSB first. The generator polynomial is $x^{16}+x^{15}+x^2+1$.
SENT-MSB	The input is handled in little endian in the lower 4-bit units of one byte and the result of SENT-compliant CRC operation is obtained with the MSB first and an initial value of 0x5. The generator polynomial is $x^4+x^3+x^2+1$.
32-ETHERNET [V1.02 or later]	The result of CRC-32-ETHERNET operation is obtained with an initial value of 0xFFFFFFFF, inverse of XOR performed, and the bits reversed. The generator polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$.

When the specification of the operation method is omitted, it is assumed that 16-CCITT-MSB-LITTLE-4 has been specified.

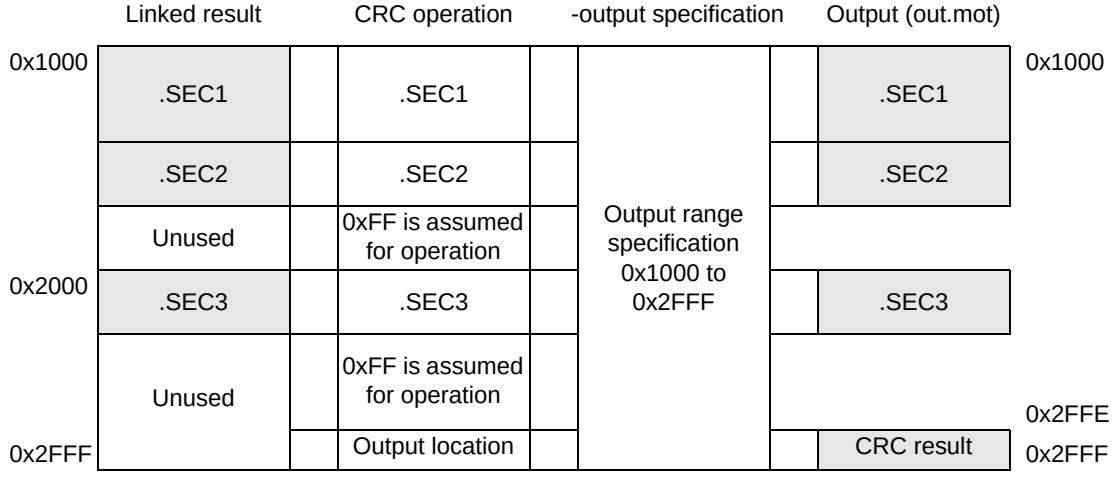
- Specify the initial value for the operation for *initial-value*.
  The range of specifiable values is from 0x0 to 0xFFFFFFFF when the operation method is 32-ETHERNET, from 0x0 to 0xF when the operation method is SENT-MSB, and from 0x0 to 0xFFFF for other cases.

  When the specification of the initial value is omitted, operation is performed on the assumption that 0x5 has been specified for the operation method of SENT-MSB, 0xFFFF for CCITT, 0xFFFFFFFF for 32-ETHERNET, and 0x0 for other cases.
- Specify the endian, size, and offset for *endian*.
  The following shows the available combinations.
  - LITTLE
  - LITTLE-2-0
  - LITTLE-4-0
  - BIG
  - BIG-2-0
  - BIG-4-0
- When the endian specification of big or little is omitted, the endian is the same as that of the input object file.
- When the operation result is output to the specified output address, data is written in the byte order specified as BIG or LITTLE at the specified offset from the beginning of the area allocated with the specified size. 0 is output from the beginning of the allocated area until immediately before the offset location.
- When the size and offset are omitted, the size is assumed to be 2 bytes and the offset is assumed to be 0.
- When the -space option is not specified, the -space=FF option is assumed for CRC operation for the unused areas in the operation range.
  Note that 0xFF is only assumed for CRC operation for the unused areas, but the areas are not actually filled with 0xFF.
  Operation is done from the lower to the higher addresses of the specified operation range.
- If this option is specified more than once, the results of all the specified CRC operations will be output. [V1.12 or later]

[Example of use]

Example 1.

```
>rlink *.obj -form=stype -start=.SEC1,.SEC2/1000,.SEC3/2000 -crc=2FFE=1000-2FFD
-output=out.mot=1000-2FFF
```



-crc option: -crc=2FFE=1000-2FFD

CRC operation is done for the area from 0x1000 to 0x2FFD and the result is output to address 0x2FFE.
When the -space option is not specified, the -space=FF option is assumed for CRC operation for the unused areas in the operation range.

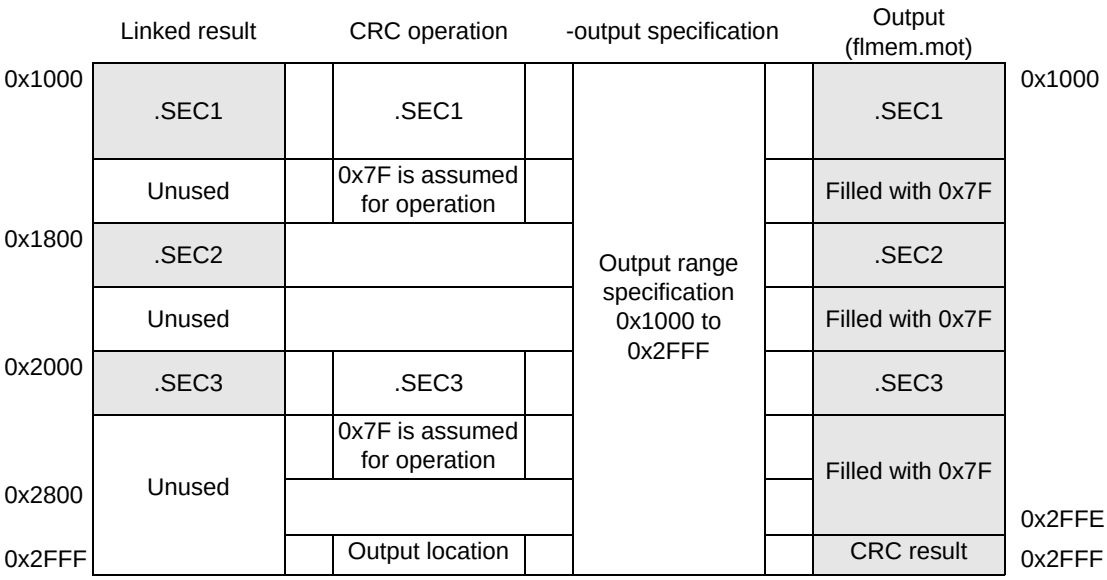
-output option: -output=out.mot=1000-2FFF
As the -space option is not specified, nothing is output to out.mot for the unused areas.
0xFF is assumed for CRC operation for the unused areas, but the areas are not actually filled with 0xFF.

Caution 1. The CRC output location cannot be included in the operation range.

Caution 2. The CRC output location should be included in the output range specified by the -output option.

Example 2.

```
>rlink *.obj -form=stype -start=.SEC1/1000,.SEC2/1800,.SEC3/2000 -space=7F
-crc=2FFE=1000-17FF,2000-27FF -output=out.mot=1000-2FFF
```



-crc option: -crc=2FFE=1000-17FD,2000-27FF
CRC operation is done for two areas from 0x1000 to 0x17FD and from 0x2000 to 0x27FF, and the result is output to address 0x2FFE.
Multiple non-contiguous operation ranges can be specified as the target of CRC operation.

-space option: -space=7F
For unused areas in the specified operation ranges, the value (0x7F) specified by the -space option is used for CRC operation.

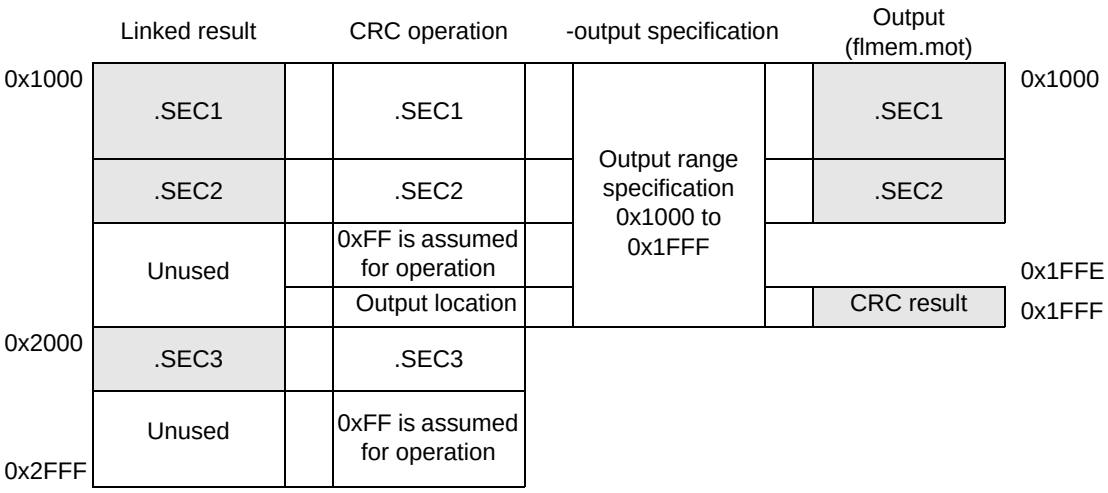
-output option: -output=out.mot=1000-2FFF
As the -space option is specified, data for the unused areas is output to out.mot.
The unused areas are filled with 0x7F.

Caution 1. CRC operation is not done in the order of the operation range specifications. CRC is calculated in the order from the lower to the higher addresses.

Caution 2. When both the -crc option and the -space option are specified, "random" or a value larger than 2 bytes must not be specified in the -space option. Be sure to specify a 1-byte value.

Example 3.

```
>rlink *.obj -form=stype -start=.SEC1,.SEC2/1000,.SEC3/2000
-crc=1FFE=1000-1FFD,2000-2FFF -output=flmem.mot=1000-1FFF
```



-crc option: -crc=1FFE=1000-1FFD,2000-2FFF

CRC operation is done for areas from 0x1000 to 0x1FFD and from 0x2000 to 0x2FFF, and the result is output to address 0x1FFE.

When the -space option is not specified, the -space=FF option is assumed for CRC operation for the unused areas in the operation range.

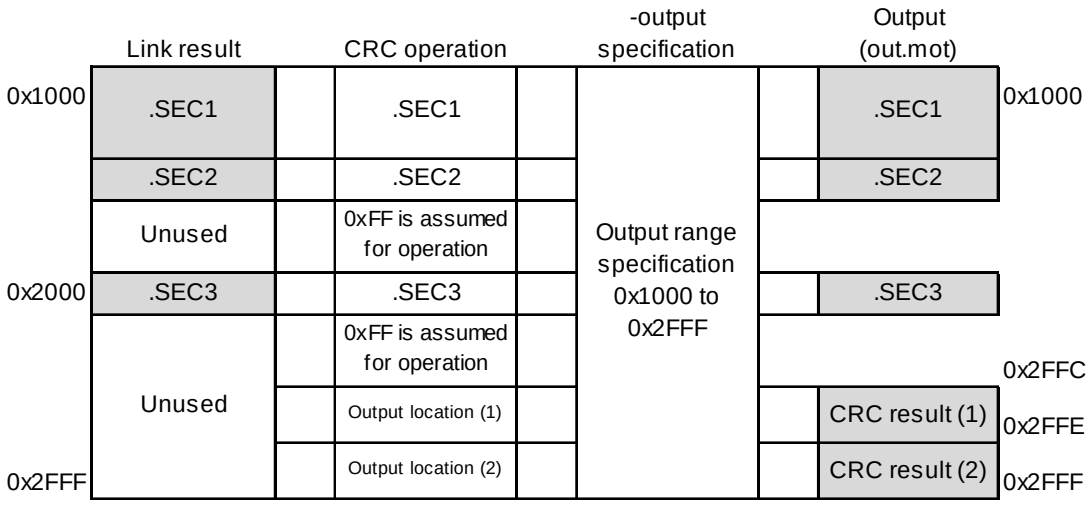
-output option: -output=flmem.mot=1000-1FFF

As the -space option is not specified, nothing is output to flmem.mot for the unused areas.

0xFF is assumed for CRC operation for the unused areas, but the areas are not actually filled with 0xFF.

Example 4.

```
>rlink *.obj -form=stype -start=.SEC1,.SEC2/1000,.SEC3/2000 -output=out.mot=1000-2FFF
-crc=2FFC=1000-1FFF -crc=2FFE=2000-2FFB
```



-crc option (1): -crc=2FFC=1000-1FFF

CRC operation is done for areas from 0x1000 to 0x1FFF, and the result is output to address 0x2FFC.

-crc option (2): -crc=2FFE=2000-2FFB

CRC operation is done for areas from 0x2000 to 0x2FFB, and the result is output to address 0x2FFE.

[Remark]

- This option does not take effect when multiple load module files are input.
- [V1.06 or earlier]
  This option is valid only when the -form={stype|hexadecimal} option is specified.
  [V1.07 or later]
  This option will be valid only if the -form={stype | hexadecimal | bin} option is specified.
- When the -space option is not specified and the operation range includes an empty area that is not output, 0xFF is assumed to be stored in the unused area during CRC operation.
- An error will occur if the CRC operation range includes an overlaid area.
- The following is CRC type mapping from OC78K0R (the object converter of the RL78,78K0R C compiler package CA78K0R (sold separately)) to the optimizing linker.

OC78K0R	Optimizing Linker
HIGH	16-CCITT-MSB-LITTLE-4
HIGH(SENT)	SENT-MSB
GENERAL	16-CCITT-LSB

-VECT

This option stores an address value in the unused areas in the vector table.

[Specification format]

```
-VECT={symbol|address}
```

- Interpretation when omitted
  None

[Detailed description]

- This option stores the specified address value in the unused areas of the vector table (addresses where no handler address value is stored).
- When this option is specified, the optimizing linker creates a vector table section and stores the specified address value in the vector table addresses even if no interrupt handlers are written in the source program.
- Specify the external name of the target function prefixed with an underscore (_) as *symbol*.
- Specify the desired hexadecimal address for *address*.
- The value to be set at address 0x2 and 0x3 of the vector table is determined in the following order of priority.
  -rrm option > -debug_monitor option > Assembly source file specification > -vectn option > -vect option

[Example of use]

- To store the address of _dummy in the unused locations in the vector table, describe as:

```
>rlink a.obj b.obj -vect=_dummy
```

[Remark]

- This option is ignored when the user creates a vector table address section in the source program because the vector table is not automatically created in this case.
- When the {*symbol*|*address*} specification is started with 0, the whole specification is assumed as an address.
- If the -form={object|relocate|library} option, -strip option, or -extract option is specified, this option will be invalid.

-VECTN

This option stores address values in the specified areas in the vector table.

[Specification format]

```
-VECTN=suboption[, ...]
  suboption := {vector-number=symbol|vector-number=address}
```

- Interpretation when omitted
  None

[Detailed description]

- This option stores the specified address values in the specified locations of the vector table.
- When this option is specified, the optimizing linker creates a vector table section and stores the specified address values in the vector table even if no interrupt handlers are written in the source program.
- Specify an even number within the range of 0x0 to 0x7E in hexadecimal for "vector-number".
- Specify the external name of the target function prefixed with an underscore (_) as *symbol*.
- Specify the desired hexadecimal address for *address*.
- The value to be set at address 0x2 and 0x3 of the vector table is determined in the following order of priority.
  -rrm option > -debug_monitor option > Assembly source file specification > -vectn option > -vect option
- When the -SPLIT_VECT option is not specified, set a value in an unused area which is not specified with the -VECTN option according to the following priority.
  1. Value specified with the VECT option
  2. If there is a defined symbol with the name (internal name) of "__dummy_int" in the link target, the address of that symbol
  3. If there is a defined symbol with the name (internal name) of "dummy_int" in the link target, the address of that symbol
  4. 0 for cases other than any of the above
- When the -SPLIT_VECT option is specified, a section for each vector number is not generated for an unused area which is not specified with the -VECTN option.

[Example of use]

- To store the address of _dummy at address 0x14 in the vector table, describe as:

```
>rlink a.obj b.obj -vectn=14=_dummy
```

[Remark]

- This option is ignored when the user creates a vector table address section in the source program because the vector table is not automatically created in this case.
- If the -form={object|relocate|library} option, -strip option, or -extract option is specified, this option will be invalid.

-SPLIT_VECT [V1.07 or later]

This option generates vector table sections split by vector table address.

[Specification format]

```
-SPLIT_VECT
```

- Interpretation when omitted
  None

[Detailed description]

- This option generates vector table sections split by vector table address.
- A vector table section is not generated for an unused area of the vector table address.

[Example of use]

- To generate a vector table section ".vect14" for vector table address 0x14, code as:

```
>rlink a.obj b.obj –vectn=14=__dummy -split_vect
```

[Remark]

- If the -vect option is specified, this option will be invalid.
- If the -form={object | relocate | library} option, -strip option, or -extract option is specified, this option will be invalid.

-VFINFO

This option outputs the variable/function information file.

[Specification format]

```
-VFINFO[=file]                    [V1.04 or earlier]
-VFINFO[=[file][(attribute,...)]] [V1.05 or later]
```

- Interpretation when omitted
  The variable/function information file is not output.

[Detailed description]

- Specify the variable/function information file as *file*.
- If the file name is omitted, vfinfo.h is output as the variable/function information file.
- When the file name has no extension, ".h" is assumed as the extension.
- callt, near, rom_forbid, or far_forbid can be specified for the output attribute attribute. A variable/function information file is generated as shown below, according to the output attribute. [V1.05 or later]

Variable Information	
Output Attribute	Description
No specification	#pragma saddr is output for frequently referenced variables for the amount of surplus space remaining in the saddr area. Variables that have already been declared as saddr variables continue to be saddr variables regardless of how many times they are referenced. #pragma saddr will not be output for those variables, and the variable information will be output with those variables commented out.
Function Information	
Output Attribute	Description
No specification or callt	#pragma callt is output for frequently called functions for the amount of surplus space remaining in the callt entry or near area. Functions that have already been declared as callt functions continue to be callt functions regardless of how many times they are called. #pragma callt will not be output for those functions, and the function information will be output with those functions commented out.
near	#pragma near is output for frequently called functions for the amount of surplus space remaining in the near area. #pragma near will not be output for functions that have already been declared as callt functions or near functions regardless of how many times they are called. The function information will be output with those functions commented out.
callt or near	#pragma callt is output for frequently called functions for the amount of surplus space remaining in the callt entry or near area. Functions that have already been declared as callt functions continue to be callt functions regardless of how many times they are called. #pragma callt will not be output for those functions, and the function information will be output with those functions commented out. Next, from among the remaining functions, #pragma near is output for frequently called functions for the amount of surplus space remaining in the near area. Functions that have already been declared as near functions continue to be near functions regardless of how many times they are called. #pragma near will not be output for those functions, and the function information will be output with those functions commented out.

rom_forbid	#pragma callt or #pragma near will not be output for functions in the section specified by the ROM option.
far_forbid	#pragma callt or #pragma near will not be output for functions in an absolute address section or a section specified as a far area by the -start option.

[Example of use]

- To output the variable/function information file to file.h, describe as:

```
>rlink a.obj b.obj -vfinfo=file.h
```

[Remark]

- If the -form={object|relocate|library} option, -strip option, or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When the section allocation address exceeds the allowable address range, information regarding only the symbols and sections allocated within the allowable areas are output to the variable/function information file. [V1.04]
- When section allocation exceeds the area specified for the object files, a warning message is output and the external variable allocation information file is output. [V1.05 or later]

-CFI [Professional Edition only] [V1.06 or later]

This option generates the function list for use in detecting illegal indirect function calls.

[Specification format]

```
-CFI
```

- Interpretation when omitted
  The function list for use in detecting illegal indirect function calls is not generated.

[Detailed description]

- This option selects generation of the function list for use in detecting illegal indirect function calls.
  For details on detecting illegal indirect function calls, refer to the item on the '-control_flow_integrity [Professional Edition only] [V1.06 or later]' compile option.
  Since the linker generates the function list for the .constf section, the .constf section must be specified with the -start option at the time of linking.
- When an object file is created with -control_flow_integrity specified at the time of compilation, the linker generates the function list according to information that the compiler has automatically extracted.
- When an object file is created without -control_flow_integrity specified at the time of compilation, the linker generates function lists for all symbols that were resolved for relocation in the object file.
- To add specific functions to the function list, specify the -CFI_ADD_Func link option.
  When a function in the specific object file is to be exempted from the function list, specify the -CFI_IGNORE_Module link option.

-CFI_ADD_Func [Professional Edition only] [V1.06 or later]

This option specifies the symbol or address of a function to be added to the function list for use in detecting illegal indirect function calls.

[Specification format]

```
-CFI_ADD_Func={symbol|address}[, ...]
```

- Interpretation when omitted
  None

[Detailed description]

- This option registers the symbol or address of functions in the function list for use in detecting illegal indirect function calls.
  For details on detecting illegal indirect function calls, refer to the item on the '-control_flow_integrity [Professional Edition only] [V1.06 or later]' compile option.
- Specify addresses in hexadecimal.
- If the specified symbol of a function is not included in the load module that was optimized by the linker, an error will occur.
- If this option is specified more than once, all specified symbols or addresses of functions are registered in the function list.
- When this option is used, the -CFI option must also be specified. If the -CFI option is not specified, an error will occur.

[Example of use]

- To register the main function of the C source code, function address 0x100, and the function sub in the C source code in the function list, write this as:

```
>rlink -cfi -cfi_add_func=_main,100 -cfi_add_func=_sub a.obj b.obj
```

-CFI_IGNORE_Module [Professional Edition only] [V1.06 or later]

This option specifies object files to be exempted from the function list for use in detecting illegal indirect function calls.

[Specification format]

```
-CFI_IGNORE_Module=suboption[, ...]
[V1.06 or earlier]
  suboption := file
[V1.07 or later]
  suboption := file[(module[, ...])]
```

- Interpretation when omitted
  None

[Detailed description]

- [V1.06 or earlier]
  This option specifies object files to be exempted from the function list for use in detecting illegal indirect function calls.
  [V1.07 or later]
  This option specifies object files and library files to be exempted from the function list for use in detecting illegal indirect function calls. The module name in a library can be used to specify a library file.
  For details on detecting illegal indirect function calls, refer to the item on the '-control_flow_integrity [Professional Edition only] [V1.06 or later]' compiler option.
- If the specified object file does not exist, an error will occur.
- If this option is specified more than once, the functions of all specified object files are exempted from the function list.
- When this option is used, the -CFI option must also be specified. If the -CFI option is not specified, an error will occur.

[Example of use]

- To remove functions in a.obj, b.obj, and c.obj from the function list, write this as:

```
>rlink -cfi -cfi_ignore_module=a.obj,b.obj -cfi_ignore_module=c.obj
```

- [V1.07 or later]
  To remove functions in the c module in the b.lib library from the function list, code as:

```
>rlink -cfi -cfi_ignore_module=b.lib(c) -lib=b.lib a.obj
```

-RAM_INIT_TABLE_SECTION [V1.12 or later]

This option generates an information table for RAM initialization.

[Specification format]

```
-RAM_INIT_TABLE_SECTION[= section-name]
```

- Interpretation when omitted
  An information table for RAM initialization will not be generated.

[Detailed description]

- This option generates an information table for RAM initialization.
- The information table is generated for the specified section name.
- If a section name is not specified, the table is generated with section name ".ram_init_table".
- Generally, the contents of the .data section are transferred from ROM to RAM at startup of a program. In addition, the RAM to which the .bss section is allocated is cleared to 0. This option generates a table containing the address and size information required for such operations.
- The information table contains a series of records that have the following fields.

	Size	Description
Field 1	4 bytes	ROM start address: For data transfer from ROM to RAM RAM start address: For clearing the RAM 0: End of the record
Field 2	2 bytes	Transfer size: For data transfer from ROM to RAM or for clearing the RAM 0: End of the record
Field 3	2 bytes	RAM start address: For data transfer from ROM to RAM or for clearing the RAM 0: End of the record

- Fields 1 to 3 are set to 0 at the end of the table.
- During startup processing, perform the following processing up to the end of the table:
  - If field 1 and field 3 are different, copy the memory data for the size of field 2 from field 1 to field 3.
  - If field 1 and field 3 are the same, clear the memory data for the size of field 2 from field 1 or field 3.
- The sections to be initialized are used for CPU program operations. Sections for the FLEXIBLE APPLICATION ACCELERATOR (FAA) are not contained in the table.

[Example of use]

- To generate an information table for RAM initialization, code as:

```
> rlink a.obj b.obj -start=.data/1000,.dataR/f1000,.bss/f2000 -rom=.data=.dataR
-ram_init_table_section=.ram_init_table
```

- If the size of the .data section is 0x100 bytes and the size of the .bss section is 0x200 bytes, the following table is generated for the .ram_init_table section.

Field 1	Field 2	Field 3
0x01000	0x100	0x1000
0xf2000	0x200	0x2000
0	0	0

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

List output

The list output options are as follows.

- -LISt
- -SHow

-LISt

This option outputs the list file.

[Specification format]

```
-LISt[=file]
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs list file *file*.
- An error will occur if the specification of the file name is omitted.

Specified Option	File Type	File Name
-form=library option or -extract option	Library list file	*Output file name*[Note].lbp
Other than above	Link map file	*Output file name*[Note].map

Note If there are two or more output files, this is the first input file name.

- Even if the section allocation address exceeds the allowable address range, this option outputs the link map information and symbol information. In this case, "**OVER**" is output. [V1.04 or later]

[Example of use]

- To output the link map file to file.map, describe as:

```
>rlink a.obj b.obj -list=file.map
```

-SHow

This option specifies information that is output to the list file.

[Specification format]

```
-SHow[=info[,info]...]
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies information *info* that is output to the list file.
- The items that can be specified as *info* are shown below.

Output Information (*info*)	When -form=library Option Is Specified	When Other Than -form=library Option Is Specified
SYmbol	Outputs symbol names within a module.	Outputs the symbol address, size, type, and optimization status.
Reference	Not specifiable	Outputs the symbol address, size, type, optimization status, and number of symbol references.
SEction	Outputs section names in a module.	Not specifiable
Xreference	Not specifiable	Outputs cross reference information.
Total_size	Not specifiable	Outputs the total sizes of sections separately for ROM-allocated sections and RAM-allocated sections.
VECTOR	Not specifiable	Outputs vector information.
STRUCT	Not specifiable	Outputs information on structure or union members (which have been defined in files that have been compiled with the -g option specified) as part of the symbol information in the link map file. This setting will be invalid if the -form=relocate/object, -rename, -lib_rename, -hide, -compress, or -optimize=symbol_delete option is specified.
RELOCATION_ATTRIBUTE	Not specifiable	Outputs the relocation attribute.
CFI	Not specifiable	When the -form=absolute option is specified Outputs the function list for use in detecting illegal indirect function calls. When the -form=hex/bin/stype option is specified and input files are other than absolute/hex/stype Outputs the function list for use in detecting illegal indirect function calls.

Output Information (*info*)	When -form=library Option Is Specified	When Other Than -form=library Option Is Specified
ALL	Outputs symbol names and section names in a module.	When the -form=relocate option is specified Outputs the same information as when the -show=symbol,xreference,total_size option is specified. When the -form=absolute option is specified Outputs the same information as when the -show=symbol,reference,xreference,total_size,vector,struct option is specified. When the -form=hexadecimal/stype/binary option Outputs the same information as when the -show=symbol,reference,xreference,total_size,vector,struct option is specified. When the -form=object option is specified Not specifiable

Remark See "3.2 Link Map File" and "3.4 Library List File" for details about output information.

- See [Remark] for details about when the specification of output information is omitted.

[Example of use]

- To output the symbol address, size, type, optimization contents, and number of symbol references, describe as:

```
>rlink a.obj b.obj -list -show=symbol,reference
```

[Remark]

- The following table shows whether output information *info* will be valid or invalid by the combinations of the -form option and the -show or -show=all option.

		SYmbol	Reference	SEction	Xreference	Total_size	VECTOR	STRUCT	RELOCATION_ATTRIBUTE	CFI
-form=absolute	only -show	Valid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Valid	Invalid	Valid	Valid	Valid	Valid	Invalid	Invalid
-form=library	only -show	Valid	Invalid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Invalid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
-form=relocate	only -show	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Invalid	Invalid	Valid	Invalid	Valid	Invalid	Invalid	Invalid
-form=object	only -show	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
-form=hexadecimal[Note 2]/stype[Note 3]/binary	only -show	Valid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Valid	Invalid	Valid	Valid[Note1]	Valid[Note1]	Valid[Note1]	Invalid	Invalid

Note 1. If a load module file, Intel Hex file, or Motorola S-record file is input, this combination will be invalid.

Note 2. If an Intel Hex file is input, the -show option cannot be specified.

Note 3. If a Motorola S-record file is input, the -show option cannot be specified.

The limitations on the output of the cross reference information are shown below.

- When a load module file is input, the referrer address information is not output.
- The information about references to constant symbols within the same file is not output.
- When optimization is specified during compilation, information about branches to immediate subordinate functions is not output.

Optimization

The optimization options are as follows.

- -Optimize
- -NOOptimize
- -SEction_forbid
- -Absolute_forbid
- -SYmbol_forbid [V1.02 or later]
- -ALLOW_OPTIMIZE_ENTRY_BLOCK [V1.13 or later]

-Optimize

This option specifies whether to execute inter-module optimization.

[Specification format]

```
-OPtimize[=Branch] [V1.01]
-Optimize[={SYmbol_delete|Branch|SPeed|SAFe}] [V1.02 or later]
```

- Interpretation when omitted
  All optimizations are provided. It is the same result as when the -optimize option is specified.

[Detailed description]

- Optimization is applied to the files specified by the -goptimize option at compilation or assembly.
- Whether to execute each type of optimization is specified by using the suboptions.

Parameter	Meaning	Program To Be Optimized	
		C Language	Assembly Language
None	Provides all optimizations.	OK	NG
symbol_delete [V1.02 or later]	Deletes variables or functions that have not been referenced even once. Be sure to specify the entry option at compilation.	OK	NG
branch	Optimizes branch instruction size according to program allocation information. Even if this option is not specified, it is performed when any other optimization is executed.	OK	OK
speed [V1.02 or later]	Executes types of optimization except for those that may slow down the object speed. Same as -optimize=symbol_delete or -optimize=branch.	OK	NG
safe [V1.02 or later]	Executes types of optimization except for those that may restrict the attributes of variables or functions. Same as -optimize=branch.	OK	NG

[Example of use]

- To optimize the branch size, describe as:

```
>rlink a.obj b.obj -optimize=branch
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- When the execution address (entry) is not specified, -optimize=symbol_delete is invalid.
- If an invalid parameter is specified, a warning will be output and the specification will be ignored.

-NOOptimize

This option disables inter-module optimization.

[Specification format]

```
-NOOPtimize
```

- Interpretation when omitted
  It is the same result as when the -optimize option is specified.

[Detailed description]

- This option disables inter-module optimization.

[Example of use]

- To disable inter-module optimization, describe as:

```
>rlink a.obj b.obj -nooptimize
```

-SEction_forbid

This option disables optimization for the specified section.

[Specification format]

```
-SEction_forbid=sub[,sub]...
    sub: [file-name|module-name](section-name[,section-name]...)
```

- Interpretation when omitted
  Optimization for the specified section is not disabled.

[Detailed description]

- This option disables optimization for the specified section.
- Specify the file name, module name, and section name for *sub*.
- If an input file name or library module name is also specified, the optimization can be disabled for a specific file, not only the entire section.

[Example of use]

- To disable all optimizations for the .SEC1 section, describe as:

```
>rlink a.obj b.obj -optimize -section_forbid=(.SEC1)
```

- To disable all optimizations for the .SEC1 and .SEC2 sections in a.obj. describe as:

```
>rlink a.obj b.obj -optimize -section_forbid=a.obj(.SEC1,.SEC2)
```

[Remark]

- This option is ignored if optimization is not applied at linkage.
- To disable optimization for an input file with its path name, type the path with the file name.
- To disable optimization for the mirror destination area, specify the mirror source area.

-Absolute_forbid

This option disables optimization regarding address + size specification.

[Specification format]

```
-Absolute_forbid=address[+size][,address[+size]]...
```

- Interpretation when omitted
  Optimization regarding address + size specification is not disabled.

[Detailed description]

- This option disables optimization regarding address + size specification.
- Specify the hexadecimal address and size for *address* and *size*.

[Caution]

- In the allocation of a section to memory before optimization, if some sections are allocated to overlapping areas and the addresses of those areas are specified with optimization disabled (absolute_forbid), optimization is not applied to all of the overlapping sections. Accordingly, optimization is unexpectedly disabled, which causes overlaps or overflows in section allocation.
  In such cases, optimization must not be disabled by specifying an address (absolute_forbid) but disabled by specifying the section (section_forbid).

[Example of use]

- To disable optimization for addresses 0x1000 to 0x11ff, describe as:

```
>rlink a.obj b.obj -optimize -absolute_forbid=1000+200
```

[Remark]

- This option is ignored if optimization is not applied at linkage.

-SYmbol_forbid [V1.02 or later]

This option disables optimization regarding deletion of unreferenced symbols.

[Specification format]

```
-SYmbol_forbid=symbol-name[,symbol-name]...
```

- Interpretation when omitted
  Optimization regarding deletion of unreferenced symbols is not disabled.

[Detailed description]

- This option disables optimization regarding deletion of unreferenced symbols.

[Example of use]

- To disable optimization of the _func symbol, describe as:

```
>rlink a.obj b.obj –optimize –symbol_forbid=_func
```

[Remark]

This option is ignored if optimization is not applied at linkage.

-ALLOW_OPTIMIZE_ENTRY_BLOCK [V1.13 or later]

This option performs optimization on the areas that are allocated before the execution start symbol.

[Specification format]

```
-ALLOW_OPTIMIZE_ENTRY_BLOCK
```

- Interpretation when omitted
  Optimization is not performed on any area allocated before the execution start symbol.

[Detailed description]

- This option performs optimization on the areas that are allocated before the execution start symbol.
- Specifying this option more than once has the same effect as specifying it once only. A warning is output in this case.

[Example of use]

- To perform optimization including the areas that are allocated before the execution start symbol, describe as:

```
>rlink a.obj b.obj -optimize -entry=_main -allow_optimize_entry_block
```

[Remark]

- This option is invalid for link processing that does not use optimization.
- If an address is specified by the -entry option, this option outputs a warning and ignores the specification.
- This option is invalid unless the -entry option is specified.

Section specification

The section specification options are as follows.

- -STARt
- -FSymbol
- -USER_OPT_BYTE
- -OCDBG
- -SECURITY_OPT_BYTE [V1.12 or later]
- -SECURITY_ID
- -FLASH_SECURITY_ID [V1.12 or later]
- -AUTO_SECTION_LAYOUT
- -SPLIT_SECTION [V1.12 or later]
- -STRIDE_DSP_MEMORY_AREA [V1.12 or later]
- -DEBUG_MONITOR
- -RRM
- -SELF
- -SELFW
- -OCDTR
- -OCDTRW
- -OCDHPI
- -OCDHPIW
- -DSP_MEMORY_AREA [V1.12 or later]

-STARt

This option specifies the start address of the section.

[Specification format]

```
-STARt=suboption[, ...]
  suboption := section-group[/address]
  section-group := section-list[: ...]
  section-list := section[, ...]
```

- Interpretation when omitted
  Absolute address sections are allocated from smallest to largest, and then relative address sections starting at the end of the absolute address sections are allocated, in the order of appearance of the input files.

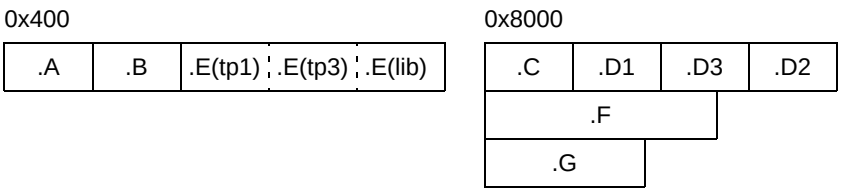
[Detailed description]

- This option specifies start address *address* of section *section*.
  Specify *address* in hexadecimal.
- Wildcard characters (*, ?) can also be used for *section*.
  The section specified with wildcard characters are expanded in the input order.
- Two or more sections (specifing by delimiting them with a comma (,)) can be allocated to the same address (i.e., sections are overlaid) by delimiting them with a colon (:).
  Sections specified at a single address are allocated in their specified order.
  Sections to be overlaid can be changed by enclosing them by parentheses "()".
- Objects in a single section are allocated in the specified order of the input file and the input library.
- If the specification of an address is omitted, the section is allocated from address 0.
- A section that is not specified by the -start option is allocated after the last allocation address.

[Example of use]

- The example below shows how sections are allocated when the objects are input in the following order (The names enclosed by parentheses are sections in each object).
  tp1.obj(.A,.D1,.E)
  tp2.obj(.B,.D3,.F)
  tp3.obj(.C,.D2,.E,.G)
  lib.lib(.E)

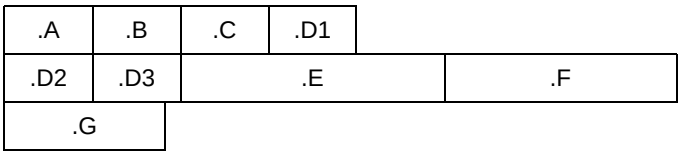
  - When the -start=.A,.B,.E/400,.C,.D*:.F:.G/8000 option is specified



  - Sections .C, .F, and .G delimited by ":" are allocated to the same address.
  - Sections specified with wildcard characters (in this example, the sections whose names start with ".D") are allocated in the input order.
  - Objects in the sections having the same name (section .E in this example) are allocated in the input order.
  - An input library's sections having the same name (section .E in this example) are allocated after the input objects.

- When the -start=.A,.B,.C,.D1:.D2,.D3,.E,.F:.G/400 option is specified

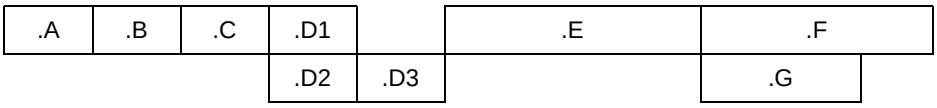
0x400



  - The sections that come immediately after ":" (sections .A, .D2, and .G in this example) are selected as the start and allocated to the same address.

- When the -start=.A,.B,.C,(.D1:.D2,.D3),.E,(.F:.G)/400 option is specified

0x400



  - When the sections to be allocated to the same address are enclosed by "()", the sections within "()" are allocated to the address immediately after the sections that come before the "()" (sections .C and .E in this example).
  - The section that comes after the "()" (section .E in this example) is allocated after the last of the sections enclosed by "()".

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- "()" cannot be nested.
- One or more colons must be described within "()".
  If ":" is not described, "()" cannot be described.
- If "()" is described, ":" cannot be described outside of "()".
- When "()" is used in this option, the optimization function of the linker is disabled.

-FSymbol

This option outputs external defined symbols to the symbol address file.

[Specification format]

```
-FSymbol=section[,section]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs the external defined symbols in section *section* to a file (symbol address file) in the form of assembler directives.
  The file name is "*output file name*.fsy".

[Example of use]

- To output the external defined symbols in sections ".A" and ".B" to symbol address file "test.fsy", describe as:

```
>rlink a.obj b.obj -fsymbol=.A,.B -output=test.abs
```

The output example of symbol address file "test.fsy" is shown below.

```
;RENESAS OPTIMIZING LINKER GENERATED FILE XXXX.XX.XX
        ;fsymbol = .A, .B

        ;SECTION NAME = .A
            .public _f
        _f .equ 0x00000000
            .public _g
        _g .equ 0x00000010
        ;SECTION NAME = .B
            .public _main
        _main .equ 0x00000020
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

-USER_OPT_BYTE

This option specifies the value set for the user option bytes.

[Specification format]

```
-USER_OPT_BYTE=user-option-byte
```

- Interpretation when omitted
  When this option is not used, be sure to set the user option byte value by using an assembly source file.

[Detailed description]

- This option specifies the value set for the user option bytes for *user-option-byte*.
- The user option byte value depends on the device in use. See the user's manual of the device for the value to be specified.
  When this option is omitted, the user option bytes are set to the initial value specified in the device file.
- Specify a hexadecimal value from 0x0 to 0xFFFFFF for the user option bytes.
  If the specified value is less than 3 bytes, the higher bits are filled with 0.
- The user option bytes is specified at addresses 0xC0 to 0xC2.
  The specified value is stored in byte units from the MSB side of the user option bytes in the order from 0xC0 to 0xC2.
- The user option byte value to be allocated at addresses 0xC0 to 0xC2 can also be specified by defining the segment with relocation attributes shown below, in the assembly source file.
  However, define the segment with 4 bytes in total, including the control value at address 0xC3.

```
        .section    .option_byte, opt_byte
          .db       0xFD                ;Address 0xC0
          .db       0xFE                ;Address 0xC1
          .db       0xFF                ;Address 0xC2
          .db       0x04                ;Address 0xC3
```

- If specification of the device file and specification of this option are made in duplicate, this option takes priority.
- If specification of the assembly source file and specification of this option are made in duplicate, this option takes priority.
- If a device file is specified in addition to the specification of the user option bytes in the assembly source file, the specification in the assembly source file takes priority.
- If a device file is specified but the user option bytes are not specified in the assembly source file or through this option, a warning is output to notify that the initial value in the device file is used.

[Example of use]

- To specify 0xFD at address 0xC0, 0xFE at address 0xC1, and 0xFF at address 0xC2 as the user option byte value, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -user_opt_byte=FDFEFF
```

[Remark]

- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When specifying the user option byte value in the assembly source file, do not use a label reference. If such an attempt is made, the result may be an unexpected value due to relocation resolution.

-OCDBG

This option specifies the control value for the on-chip debug.

[Specification format]

```
-OCDBG=value
```

- Interpretation when omitted
  When this option is not used, be sure to set the control value for the on-chip debug by using an assembly source file.

[Detailed description]

- This option specifies the control value for the on-chip debug for *value*.
- The control value for the on-chip debug depends on the device in use. See the user's manual of the device for the value to be specified.
  When this option is omitted, the control value for the on-chip debug are set to the initial value specified in the device file.
- Specify a hexadecimal value from 0x0 to 0xFF for the control value for the on-chip debug.
- An error occurs if a value that cannot be specified for the control value for the on-chip debug is specified.
- The control value for the on-chip debug is specified at addresses 0xC3.
- The control value for the on-chip debug can also be specified by defining the segment with relocation attributes shown below, in the assembly source file. However, define the segment with 4 bytes in total, including the user option bytes starting from address 0xC0.
  If a device file is specified in addition to the specification of the control value in the assembly source file, the specification in the assembly source file takes priority.

```
        .section    .option_byte, opt_byte
          .db       0xfd                ;Address 0xC0
          .db       0xfe                ;Address 0xC1
          .db       0xff                ;Address 0xC2
          .db       0x04                ;Address 0xC3
```

- If specification of the device file and specification of this option are made in duplicate, this option takes priority.
- If the control value is specified both in the assembly source file and through this option, this option takes priority.

[Example of use]

- To set 0x04 at address 0xC3 as the control value for the on-chip debug, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdbg=04
```

[Remark]

- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When specifying the control value for the on-chip debug in the assembly source file, do not use a label reference. If such an attempt is made, the result may be an unexpected value due to relocation resolution.

-SECURITY_OPT_BYTE [V1.12 or later]

This option specifies the control value for the security option byte.

[Specification format]

```
-SECURITY_OPT_BYTE=value
```

- Interpretation when omitted
  The value specified in an assembly source file or the value defined in the device file is set.

[Detailed description]

- This option specifies the control value for the security option byte for *value*.
- Specify a hexadecimal value in the range from 0x0 to 0xFF for the control value for the security option byte.
- Specifying a value that cannot be specified for the control value for the security option byte will cause an error.
- The control value for the security option byte is specified at address 0xC4.
- The control value for the security option byte can also be specified by defining the segment with the following relocation attributes in the assembly source file.
  Define the segment with 5 bytes in total, including the user option bytes and on-chip debugging control value starting from address 0xC0.

```
        .section    .option_byte, opt_byte
          .db       0xfd                ;Address 0xC0
          .db       0xfe                ;Address 0xC1
          .db       0xff                ;Address 0xC2
          .db       0x04                ;Address 0xC3
          .db       0xff                ;Address 0xC4
```

- If the assembly source file and this option are specified at the same time, this option takes priority.
- If the device file, the assembly source file, and this option are specified at the same time, this option takes priority.

[Example of use]

- To set 0xC4 at address 0x04 as the control value for the security option byte, describe as:

```
>rlink a.obj b.obj -device=dr7f124fgj.dvf -security_opt_byte=04
```

[Remark]

- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When specifying the security ID value in the assembly source file, do not use a label reference. Doing so might result in an unexpected value due to relocation resolution.

-SECURITY_ID

This option specifies a security ID value.

[Specification format]

```
-SECURITY_ID=value
```

- Interpretation when omitted
  A security ID value is not set.

[Detailed description]

- This option specifies a security ID value for *value*.
- Specify a hexadecimal value for the security ID value.
- An error occurs if a value that cannot be specified for the security ID value is specified.
  The value is set in byte units from the MSB side of the user option byte in the order from the lower to the higher addresses.
- The location where the security ID value is stored and the maximum size of the security ID are specified in the device file.
- (V1.11 or later) Specify the value for the option that does not exceed the maximum size of the security ID of the device specifications.
  (V1.10 or earlier) Specify the value within 1 byte.
  An error occurs if the specified value exceeds the maximum size of the security ID of the device specifications.
  If the specified security ID size is less than the maximum size, the higher bits are filled with 0.
- A security ID value can also be specified by defining the segment with relocation attributes shown below, in the assembly source file.
  However, in the same way as the option specification, make sure that the defined value conforms to the security ID size of the device specifications.

```
        .section    .security_id, SECUR_ID
          .db       0x01            ;Address 0xC4
          .db       0x02            ;Address 0xC5
          .db       0x03            ;Address 0xC6
          .db       0x04            ;Address 0xC7
          .db       0x05            ;Address 0xC8
          .db       0x06            ;Address 0xC9
          .db       0x07            ;Address 0xCA
          .db       0x08            ;Address 0xCB
          .db       0x09            ;Address 0xCC
          .db       0x0A            ;Address 0xCD
```

- If the security ID value is specified both in the assembly source file and through this option, this option takes priority.
- Be sure to see the user's manual of the device to specify the security ID value.

[Example of use]

- To specify storage of the security ID values 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, and 0x0A at addresses starting from 0xC4, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -security_id=0102030405060708090A
```

[Remark]

- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When specifying the security ID value in the assembly source file, do not use a label reference. If such an attempt is made, the result may be an unexpected value due to relocation resolution.

-FLASH_SECURITY_ID [V1.12 or later]

This option specifies the value to be set for the flash programmer security ID.

[Specification format]

```
-FLASH_SECURITY_ID=value
```

- Interpretation when omitted
  A flash programmer security ID value is not set.

[Detailed description]

- This option specifies the flash programmer security ID value for *value*.
- Specify a hexadecimal value for the flash programmer security ID value.
- Specifying a value that cannot be specified for the flash programmer security ID value will cause an error.
  The value is set in byte units from the MSB side of the flash programmer security ID value in the order from lower to higher addresses.
- The location where the flash programmer security ID value is stored and the maximum size of the security ID are specified in the device file.
- Specify the value for the option that does not exceed the maximum size of the security ID of the device specifications.
  An error occurs if the specified value exceeds the maximum size of the security ID of the device specifications.
  If the size of the specified security ID is less than the maximum size, the higher bits are filled with 0.
- The security ID value can also be specified by defining the segment with the following relocation attributes in the assembly source file.
  However, in the same way as the option specification, make sure that the defined value does not exceed the maximum size of the security ID of the device specifications.

```
        .section  .flash_security_id, FLASH_SECUR_ID
          .db       0x01            ;Address 0xD6
          .db       0x02            ;Address 0xD7
          .db       0x03            ;Address 0xD8
          .db       0x04            ;Address 0xD9
          .db       0x05            ;Address 0xDA
          .db       0x06            ;Address 0xDB
          .db       0x07            ;Address 0xDC
          .db       0x08            ;Address 0xDD
          .db       0x09            ;Address 0xDE
          .db       0x0A            ;Address 0xDF
          .db       0x0B            ;Address 0xE0
          .db       0x0C            ;Address 0xE1
          .db       0x0D            ;Address 0xE2
          .db       0x0E            ;Address 0xE3
          .db       0x0F            ;Address 0xE4
          .db       0x10            ;Address 0xE5
```

- If the assembly source file and this option are specified at the same time, this option takes priority.
- Be sure to specify the security ID value by referring to the user's manual of the device.

[Example of use]

- To specify the storage of the security ID values 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, and 0x10 at the address starting from 0xD6, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -security_id=0102030405060708090A0B0C0D0E0F10
```

[Remark]

- If the -form={object|relocate|library} or -extract option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.
- When specifying the security ID value in the assembly source file, do not use a label reference. Doing so might result in an unexpected value due to relocation resolution.

-AUTO_SECTION_LAYOUT

This option automatically allocates sections.

[Specification format]

```
-AUTO_SECTION_LAYOUT
```

- Interpretation when omitted
  Sections are not automatically allocated.

[Detailed description]

- Sections are automatically allocated based on the information of the device file.
- After the section specified by the -start option and the absolute address sections have been allocated, the remaining sections are automatically allocated based on the information of the device file.
- With the -cpu option specified, -AUTO_SECTION_LAYOUT automatically allocates sections to an address in the range from *address1* to *address2* designated by the -cpu option and within the memory space defined in the device file designated by the -device option.

[Example of use]

- To automatically allocate sections from the information of the device file, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -auto_section_layout
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-SPLIT_SECTION [V1.12 or later]

This option enables automatic allocation of a section for each module.

[Specification format]

```
-SPLIT_SECTION
```

- Interpretation when omitted
  Automatic section allocation is performed in units of combined sections.

[Detailed description]

- When SPLIT_SECTION is specified, sections separated by a module are separately allocated to available areas in the device.
  When sections are automatically allocated without SPLIT_SECTION specified, the same sections of modules to be linked combine as one section, and then the combined section is allocated to an available area in the device.

  Suppose that file1.obj and file2.obj each contain text_section and data_section as shown below.

  file1.obj

text_section size 1 K
data_section size 1 K

  file2.obj

text_section size 1 K
data_section size 1 K

When the device is configured with ROM size 3 K, RAM-A size 1.5 K, and RAM-B size 1.5 K, automatic allocation of sections by using the auto_section_layout option is as follows.
The ROM size (3 K) is sufficient enough to allocate text_section of both file1.obj and file2.obj. The RAM is split to RAM-A (size 1.5 K) and RAM-B (size 1.5 K), and data_section (size 2 K) of the combined file1.obj and file2.obj cannot be allocated, as shown below.

Memory	Section allocation
ROM size 3 K	Section text_section size 2 K file1.obj: text_section size 1 K file2.obj: text_section size 1 K
	Unused size 1 K
	Memory unallocated
RAM-A size 1.5 K	Section data_section size 2 K file1.obj: data_section file2.obj: data_section
	Allocation fails with overflow of 0.5 K.
	Memory unallocated
RAM-B size 1.5 K	Unused size 1.5 K

When SPLIT_SECTION is specified, data_section (size 1 K) of file1.obj is allocated to RAM-A (size 1.5 K), and then data_section (size 1 K) of file2.obj is allocated to RAM-B (size 1.5 K), as shown below.

Memory	Section allocation
ROM size 3 K	Section text_section size 1 K file1.obj: text_section size 1 K
	Section $text_section_part01 size 1 K file2.obj: text_section size 1 K
	Unused size 1 K
	Memory unallocated
RAM-A size 1.5 K	Section data_section size 1 K file1.obj: data_section size 1 K
	Unused size 0.5 K
	Memory unallocated
RAM-B size 1.5 K	Section $data_section_part01 size 1 K file2.obj: data_section size 1 K
	Unused size 0.5 K

A section for each module has the following section name.

```
$sss_part??
```

sss : Section name before the split
?? : 01 to 99

[Example of use]

- To allocate sections for each module, describe as:

```
>rlink a.obj b.obj -device=dr5f100pj.dvf -auto_section_layout -split_section
```

[Remark]

- If the -form={object|relocate|library} or -strip option is specified, this option will be invalid.
- If the -auto_section_layout option is not specified, this option will be invalid.

-STRIDE_DSP_MEMORY_AREA [V1.12 or later]

This option enables automatic allocation of sections to areas split by the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).

[Specification format]

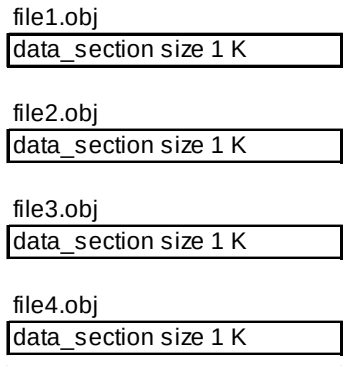
```
-STRIDE_DSP_MEMORY_AREA
```

- Interpretation when omitted
  Automatic allocation of sections is not split by the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).

[Detailed description]

- With STRIDE_DSP_MEMORY_AREA specified, sections are allocated in sequence up to just before the FLEXIBLE APPLICATION ACCELERATOR (FAA) memory area, and the remaining sections are allocated as another section from after the FLEXIBLE APPLICATION ACCELERATOR (FAA) memory area.

  Suppose that file1.obj, file2.obj, file3.obj, and file4.obj each contain data_section, as shown below.



When the device is configured with RAM-A size 2.5 K and RAM-B size 2.5 K, automatic allocation of sections by using the auto_section_layout option is as follows.
The RAM is split to RAM-A (size 2.5 K) and RAM-B (size 2.5 K), and data_section (size 4 K) of the combined file1.obj, file2.obj, file3.obj, and file4.obj cannot be allocated as shown below.

Memory	Section allocation
RAM-A size 2.5 K	Section data_section: Size 4 K file1.obj: data_section size 1 K file2.obj: data_section size 1 K file3.obj: data_section size 1 K
Memory area for DSP	file4.obj: data_section size 1 K
	Allocation fails with overflow of 1.5 K.
RAM-B size 2.5 K	Unused size 2.5 K

When STRIDE_DSP_MEMORY_AREA is specified, data_section (size 2 K) of file1.obj and file2.obj is allocated to RAM-A (size 2.5 K), and then data_section (size 2 K) of file3.obj and file4.obj is allocated to RAM-B (size 2.5 K), as shown below.

Memory	Section allocation
RAM-A size 2.5 K	Section data_section: Size 2 K file1.obj: data_section size 1 K file2.obj: data_section size 1 K
	Unused size 0.5 K
Memory area for DSP	
RAM-B size 2.5 K	Section $data_section_part01: Size 2 K file3.obj: data_section size 1 K file4.obj: data_section size 1 K
	Unused size 0.5 K

At this time, the other section allocated after the split has the following section name.

```
$sss_part??
```

sss : Section name before the split
?? : 01 to 99

[Example of use]

- To allocate sections to areas split by the FLEXIBLE APPLICATION ACCELERATOR (FAA) memory area, describe as:

```
> rlink a.obj b.obj -device=dr5f10y14.dvf -auto_section_layout -dsp_memory_area
-stride_dsp_memory_area
```

[Remark]

- If the -form={object|relocate|library} or -strip option is specified, this option will be invalid.
- When split_section is specified, this option outputs an error message and terminates execution.
- When auto_section_layout is not specified, a warning will be output and the specification will be ignored.
- When dsp_memory_area is not specified, a warning will be output and the specification will be ignored.

-DEBUG_MONITOR

This option specifies the memory area for the OCD monitor.

[Specification format]

```
-DEBUG_MONITOR[=address1-address2]
```

- Interpretation when omitted
  The memory area for the OCD monitor is not allocated.

[Detailed description]

- Specify the start address for the OCD monitor as *address1* and the end address for the OCD monitor as *address2*.
- When the start address for the OCD monitor and the end address for the OCD monitor are omitted, the following is assumed to be specified.
  - For an 8-bit CPU
    0 bytes
  - For other than an 8-bit CPU
    Start address for OCD monitor: End address for on-chip ROM - 512 + 1
    End address for OCD monitor: End address for on-chip ROM
- [V1.10 or earlier] This option fills addresses 0x2, 0x3, and 0xCE to 0xD7 and the area from the OCD monitor start address to the OCD monitor end address with 0xFF.
- [V1.11 or later] This option fills addresses 0x2 and 0x3, the area conforming to device specifications, and the area from the OCD monitor start address to the OCD monitor end address with 0xFF.
- The area of addresses 0x2 and 0x3 which is also the vector table gives priority to the following specifications.
  -rrm option > -debug_monitor option > Assembly source file specification > -vectn option > -vect option
- When the memory area for the OCD monitor includes the location to allocate the user option bytes, the following specifications are given priority.
  -user_opt_byte option > Assembly source file specification > Device file specification > -debug_monitor option
- When the memory area for the OCD monitor includes the location to allocate the control value for on-chip debugging, the following specifications are given priority.
  -ocdbg option > Assembly source file specification > Device file specification > -debug_monitor option
- When the memory area for the OCD monitor includes the location to allocate the security ID value, the following specifications are given priority.
  -security_id option > Assembly source file specification > Device file specification > -debug_monitor option

[Example of use]

- To specify addresses 0x300 to 0x3FF as the memory area for the OCD monitor, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -debug_monitor=300-3FF
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-RRM

This option specifies the work area for the RRM/DMM function.

[Specification format]

```
-RRM=address
```

- Interpretation when omitted
  The work area for the RRM/DMM function is not allocated.

[Detailed description]

- Specify the start address for the work area for the RRM/DMM function as *address*.
- Four bytes from the start address become the work area for the RRM/DMM function.
- The start address of the RRM is set to addresses 0x2 and 0x3 which are allocated by the -debug_monitor option.
- [V1.13 or later] No section is allocated to the first four bytes from the RRM start address.
- The area of addresses 0x2 and 0x3 which is also the vector table gives priority to the following specifications.
  -rrm option > -debug_monitor option > Assembly source file specification > -vectn option > -vect option

[Example of use]

- To specify four bytes from address 0xFFDE0 as the work area for the RRM/DMM function, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -debug_monitor=2FC00-2FFF -rrm=FFDE0
```

[Remark]

- This option will be invalid in any one of the following cases.
  - If the -form={object|relocate|library} option or -strip option is specified
  - If the device specified by the -device option does not support the RRM/DMM function
  - If the -debug_monitor option is not specified

-SELF

This option disables allocation of a section to the self RAM area.

[Specification format]

```
-SELF
```

- Interpretation when omitted
  It is possible to allocate a section to the self RAM area.

[Detailed description]

- This option disables allocation of a section to the self RAM area.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To disable allocation of a section to the self RAM area, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -self
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-SELFW

This option outputs a warning message when a section is allocated to the self RAM area.

[Specification format]

```
-SELFW
```

- Interpretation when omitted
  The self RAM area is not checked.

[Detailed description]

- This option outputs a warning message when a section is allocated to the self RAM area.
- When an allocated section exceeds the self RAM area, an error will occur.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To output a warning message when a section is allocated to the self RAM area, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -selfw
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-OCDTR

This option disables allocation of a section to the trace RAM and self RAM areas.

[Specification format]

```
-OCDTR
```

- Interpretation when omitted
  It is possible to allocate a section to the trace RAM and self RAM areas.

[Detailed description]

- This option disables allocation of a section to the trace RAM and self RAM areas.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To disable allocation of a section to the trace RAM and self RAM areas, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdtr
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-OCDTRW

This option outputs a warning message when a section is allocated to the trace RAM and self RAM areas.

[Specification format]

```
-OCDTRW
```

- Interpretation when omitted
  The trace RAM and self RAM areas are not checked.

[Detailed description]

- This option outputs a warning message when a section is allocated to the trace RAM and self RAM areas.
- When an allocated section exceeds the trace RAM and self RAM areas, an error will occur.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To output a warning message when a section is allocated to the trace RAM and self RAM areas, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdtrw
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-OCDHPI

This option disables allocation of a section to the hot plug-in RAM, trace RAM, and self RAM areas.

[Specification format]

```
-OCDHPI
```

- Interpretation when omitted
  It is possible to allocate a section to the hot plug-in RAM, trace RAM, and self RAM areas.

[Detailed description]

- This option disables allocation of a section to the hot plug-in RAM, trace RAM, and self RAM areas.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To disable allocation of a section to the hot plug-in RAM, trace RAM, and self RAM areas, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdhpi
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-OCDHPIW

This option outputs a warning message when a section is allocated to the hot plug-in RAM, trace RAM, and self RAM areas.

[Specification format]

```
-OCDHPIW
```

- Interpretation when omitted
  The hot plug-in RAM, trace RAM, and self RAM areas are not checked.

[Detailed description]

- This option outputs a warning message when a section is allocated to the hot plug-in RAM, trace RAM, and self RAM areas.
- When an allocated section exceeds the hot plug-in RAM, trace RAM, and self RAM areas, an error will occur.
- The __STACK_ADDR_START symbol and __STACK_ADDR_END symbol are set except for in the saddr area.

[Example of use]

- To output a warning message when a section is allocated to the hot plug-in RAM, trace RAM, and self RAM areas, describe as:

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdhpiw
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

-DSP_MEMORY_AREA [V1.12 or later]

This option disables allocation of a section to the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).

[Specification format]

```
-dsp_memory_area
```

- Interpretation when omitted
  It is possible to allocate a section to the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).

[Detailed description]

- This option disables allocation of a section to the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA).

[Example of use]

- To disable allocation of a section to the memory area shared with the FLEXIBLE APPLICATION ACCELERATOR (FAA), describe as:

```
>rlink a.obj b.obj -device=dr5f100pj.dvf -auto_section_layout -dsp_memory_area
```

[Remark]

- If the -form={object|relocate|library} or -strip option is specified, this option will be invalid.
- If the -device option is not specified, this option will be invalid.

Verify specification

The verify specification options are as follows.

- -CPu
- -CHECK_DEVICE
- -CHECK_64K_ONLY
- -NO_CHECK_SECTION_LAYOUT
- -CHECK_OUTPUT_ROM_AREA [V1.07 or later]

-CPu

This option checks the consistency of the address to which the section is allocated.

[Specification format]

```
-CPU=suboption[, ...]
  suboption := type=address1-address2
```

- Interpretation when omitted
  The consistency of the address to which the specified section is allocated is not checked.

[Detailed description]

- This option checks the consistency of the address to which the section is allocated.
  An error will be output if the section allocation address for memory type *type* does not fit in the specified address range.
- The items that can be specified as *type* are shown below.
  An error will occur if any other item is specified.

ROm	Allocates the section to a ROM area.
RAm	Allocates the section to a RAM area.
FIX	Allocates the section to a fixed-address area (e.g. I/O area). If the address range overlaps with ROM or RAM, the setting for FIX is valid.

- Specify the start address and end address of the address range to check for consistency in hexadecimal as *address1* and *address2*.

[Example of use]

- The result is normal when section .text and section .bss are respectively allocated within the ranges from 0x100 to 0x1FF and from 0x200 to 0x2FF.
  If they are not allocated within the ranges, an error will be output.

```
>rlink a.obj b.obj -start=.text/100,.bss/200 -cpu=ROM=100-1FF,RAM=200-2FF
```

[Remark]

[V1.01 or earlier]

- If the -form={object|relocate|library} option, -strip option, or -device option is specified, this option will be invalid.

[V1.02 or later]

- If the -form={object|relocate|library} option, or -strip option is specified, this option will be invalid.

-CHECK_DEVICE

This option checks the device file specified when creating an object file.

[Specification format]

```
-CHECK_DEVICE
```

- Interpretation when omitted
  The device file is not checked.

[Detailed description]

- This option checks that the device files specified when creating object files and the device file specified by the -device option are all the same.
- If there is a different device file, an error will occur.

[Example of use]

- To check that the device files specified when creating object files and the device file specified by the -device option are all the same, describe as:

```
>rlink a.obj b.obj -check_device -device=dr5f10y14.dvf
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

-CHECK_64K_ONLY

This option disables checking whether an allocated section exceeds the (64K-1)-byte boundary.

[Specification format]

```
-CHECK_64K_ONLY
```

- Interpretation when omitted
  When an allocated section exceeds the 64K-byte boundary or (64K-1)-byte boundary, an error will occur.

[Detailed description]

- This option disables checking whether an allocated section exceeds the (64K-1)-byte boundary.
- When an allocated section exceeds the 64K-byte boundary, an error will occur.
- The sections with the following relocation attributes are subject to checking whether the 64K-byte boundary or (64K-1)-byte boundary is exceeded.

Relocation Attribute	Default Section Name
TEXTF_UNIT64KP	.textf_unit64kp
CONST	.const
CONSTF	.constf
DATA	.data
BSS	.bss
DATAF	.dataf
BSSF	.bssf

- An allocated section exceeding the 64K-byte boundary means that the lower 16 bits of the section's address will exceed 0xFFFF and continue to 0x0000.
- An allocated section exceeding the (64K-1)-byte boundary means that the lower 16 bits of the section's address will exceed 0xFFFE and continue to 0xFFFF.

[Example of use]

- To disable checking whether an allocated section exceeds the (64K-1)-byte boundary, describe as:

```
>rlink a.obj b.obj -check_64k_only
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

-NO_CHECK_SECTION_LAYOUT

This option disables checking of the address to which the section is allocated.

[Specification format]

```
-NO_CHECK_SECTION_LAYOUT
```

- Interpretation when omitted
  Whether sections are allocated to memory as shown below is checked, and if they are not allocated so, an error will occur.

Section	Location for Allocation
.option_byte	Address is fixed
.security_id	Address is fixed
.sbss / .sdata	saddr area
.sbss / .sdata (RAM side when the -rom option is specified)	saddr area
.bss / .data	Internal RAM
.bss / .data (RAM side when the -rom option is specified)	Internal RAM
.const	Flash mirror space
.constf	Internal ROM
.sdata / .data (ROM side when the -rom option is specified)	Internal ROM
.text	Program memory
.text_unit64kp / .textf	Program memory

[Detailed description]

- This option disables checking whether the memory location that is read from the device file is consistent with the memory location of the section.

[Example of use]

- To disable checking of the address to which the section is allocated, describe as:

```
>rlink a.obj b.obj -no_check_section_layout
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

-CHECK_OUTPUT_ROM_AREA [V1.07 or later]

This option checks whether the output address of a HEX file ranges in internal ROM or the data flash area.

[Specification format]

```
-CHECK_OUTPUT_ROM_AREA
```

- Interpretation when omitted
  Not checked.

[Detailed description]

- If this option is specified, whether the output address of an Intel HEX file or a Motorola S-record file ranges in internal ROM or the data flash area is checked. If there is data outside of the range of internal ROM or the data flash area, a warning is output.

[Example of use]

- To check whether the address of the output data of an Intel HEX file is outside of the range of internal ROM or the data flash area, code as:

```
>rlink -form=hex a.obj b.obj -device=dr5f100pj.dvf -check_output_rom_area
```

[Remark]

- If the -device option is not specified, this option will be invalid.
- If the -form={hexadecimal | stype} option is not specified, this option will be invalid.

Subcommand file specification

The subcommand file specification option is as follows.

- -SUbcommand

-SUbcommand

This option specifies options with a subcommand file.

[Specification format]

```
-SUbcommand=file
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies options with subcommand file *file*.
- Option contents specified with a subcommand file are expanded to the location at which this option is specified on the command line and are executed.
- See "2.4.2 Subcommand file usage" for details about a subcommand file.

[Example of use]

- Create subcommand file "sub.txt" with the following content.

```
input file2.obj file3.obj       ; This is a comment.
library lib1.lib, &             ; This is a line continued.
lib2.lib
```

To specify subcommand file sub.txt, describe as:

```
>rlink file1.obj -subcommand=sub.txt file4.obj
```

The command line is expanded as follows, and the file input order is: file1.obj, file2.obj, file3.obj, file4.obj.

```
>rlink file1.obj file2.obj file3.obj -library=lib1.lib,lib2.lib file4.obj
```

Microcontroller specification

The microcontroller specification option is as follows.

- -DEVICE

-DEVICE

This option specifies the device file name used at linkage.

[Specification format]

```
-DEVICE=file
```

- Interpretation when omitted
  The information of the device file is not used at linkage.
  When this option is omitted, the settings of the user option bytes, the control value for the on-chip debug, and the security ID value become invalid.

[Detailed description]

- Specify the name of the target device file used at linkage as *file*.
- An error will occur if the specified file is not found.
- An object code corresponding to information of the specified target device file is generated.
- An error will occur if the specifications in the device file differ from the CPU core or the use of the division/multiplication and multiply-accumulate unit specified in the compiler, or the CPU core specified in the assembler.
- An error will occur if the mirror area specified in the assembler differs from that specified in the device file.

[Example of use]

- To specify target device file name DR5F10Y14.DVF, describe as:

```
>rlink file1.obj -device=dr5f10y14.dvf
```

[Remark]

- If a device file has not been specified, an error is not output even when the section for the saddr variables is located outside of the saddr area.

Other

Other options are as follows.

- -S9
- -STACk
- -COmpress
- -NOCOmpress
- -MEMory
- -REName
- -LIB_REName [V1.08 or later]
- -DELete
- -REPlace
- -EXTract
- -STRip
- -CHange_message
- -Hide
- -Total_size
- -VERBOSE [V1.10 or later]
- -LOgo
- -NOLOgo
- -END
- -EXIt

-S9

This option outputs the S9 record at the end.

[Specification format]

```
-S9
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs the S9 record at the end even if the entry point address exceeds 0x10000.

[Example of use]

- To output the S9 record at the end, describe as:

```
>rlink a.obj b.obj -form=stype -output=c.mot -s9
```

[Remark]

- If the -form=stype option is not specified, this option will be invalid.

-STACk

This option outputs the stack information file.

[Specification format]

```
-STACk
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs the stack information file.
- The file name is "*output-file-name*.sni".

[Example of use]

- To output stack information file "c.sni", describe as:

```
>rlink a.obj b.obj -output=c.abs -stack
```

[Remark]

- If the -form={object|relocate|library} option or -strip option is specified, this option will be invalid.

-COmpress

This option compresses the debug information.

[Specification format]

```
-COmpress
```

- Interpretation when omitted
  The debug information is not compressed (It is the same result as when the -nocompress option is specified).

[Detailed description]

- This option compresses the debug information.
- By compressing the debug information, the loading speed of the debugger is improved.

[Example of use]

- To compress the debug information, describe as:

```
>rlink a.obj b.obj -output=c.abs -compress
```

[Remark]

- If the -form={object|relocate|library|hexadecimal|stype|binary} option or -strip option is specified, this option will be invalid.

-NOCOmpress

This option does not compress the debug information.

[Specification format]

```
-NOCOmpress
```

- Interpretation when omitted
  The debug information is not compressed.

[Detailed description]

- This option does not compress the debug information.
- Link time when specifying this option is shorter than when the -compress option is specified.

[Example of use]

- Not to compress the debug information, describe as:

```
>rlink a.obj b.obj -output=c.abs -nocompress
```

-MEMory

This option specifies the memory size occupied during linking.

[Specification format]

```
-MEMory=[occupancy]
```

- Interpretation when omitted
  The processing is the same as when the -memory=high option is specified.

[Detailed description]

- This option specifies memory size *occupancy* occupied during linking.
- The items that can be specified as *occupancy* are shown below.

High	The optimizing linker loads the information necessary for linking in large units to prioritize the processing speed.
Low	The optimizing linker loads the information necessary for linking in smaller units to reduce the memory occupancy. This increases the frequency of file access. As a result, processing will be slower than when "High" is specified if the memory used is not larger than implementation memory.

- If *occupancy* is omitted, it is assumed that "High" has been specified.
- Specify "Low" as *occupancy* if processing is slow because a large project is linked and the memory size occupied by the optimizing linker exceeds the available memory in the machine used.

[Example of use]

- To reduce the memory occupancy, describe as:

```
>rlink a.obj b.obj -nooptimize -memory=low
```

[Remark]

- In the following cases, the specification of the -memory=low option will be invalid.
  - When the -form={absolute|hexadecimal|stype|binary} option and following options are specified at the same time
    - Any of the -compress, -delete, -rename, -lib_rename, -stack, or -optimize options
    - Combination of the -list option and the -show[={reference|xreference|struct|all}] option
  - When the -form=library option and following options are specified at the same time
    - Any of the -delete, -rename, -extract, -hide, -replace, or -allow_duplicate_module_name options
  - When the -form={object|relocate} option and following options are specified at the same time
    - -extract option

  Some combinations of this option and the input or output file format are invalid.
  See "Table 2.9 Relations Between Output Formats And Input Files Or Other Options" for details.

-REName

This option changes an external symbol name or a section name.

[Specification format]

```
-REName=suboption[, ...]
  suboption := {(names)|file(names)|module(names)}
  names := name1=name2[, ...]
```

- Interpretation when omitted
  None

[Detailed description]

- This option changes an external symbol name or a section name.
- Specify the symbol name or section name to be changed as *name1*. Specify the symbol name or section name after changing as *name2*.
- By specifying *file*, you can change only the names of the sections included in *file*.
- When the output of library files is selected (with -form=library), you can specify *module* so that only the names of the sections included in *module* within the input library will be changed.
  To change section names within the input library in other cases, use the -lib_rename option.
- By specifying *file* or *module*, you can change only the names of the global symbols included in *file* or *module*.
- When a C variable name is specified, add "_" at the head of the definition name in the program.
- If the specified name matches both section and symbol names, the symbol name is changed.
- If there are two or more files or modules with the same name, the priority depends on the input order.
- If this option is specified more than once, all specifications will be valid.
- An error will occur in the following case.
  - When the specified *name*, *file*, or *module* cannot be found

[Example of use]

- To change symbol name "_sym1" to "_data", describe as:

```
>rlink a.obj b.obj -rename=(_sym1=_data)
```

- To change section ".SEC1" in library module "lib1" to section ".SEC2", describe as:

```
>rlink -form=library -library=lib1.lib -rename=(.SEC1=.SEC2)
```

[Remark]

- If this option is specified together with the -extract option or -strip option, an error will occur.
- When the -form={absolute|hexadecimal|stype|binary} option is specified, the section name of the input library cannot be changed.
- Operation is not guaranteed if this option is used in combination with compile option -Omerge_files.

-LIB_REName [V1.08 or later]

This option changes the name of a symbol or section that was input from a library.

[Specification format]

```
-LIB_REName=suboption[, ...]
  suboption := (names)
             |  file(names)
             |  file|modules(names)
  modules := module[|module ...]
  names := name1=name2[, ...]
```

- Interpretation when omitted
  None

[Detailed description]

- This option changes the name of a global symbol or section included in a module within the library that was specified by the -library option.
- Specify the symbol name or section name to be changed as *name1*. Specify the symbol name or section name after changing as *name2*.
- When you specify a C variable name, add "_" at the head of the definition name in the program.
- If the specified name matches both section and symbol names, the symbol name is changed.
- If there are two or more files or modules with the same name, the priority depends on the input order.
- If this option is specified more than once, all specifications will be valid.
- An error will occur in any of the following cases.
  - When the specified *name*, *file*, or *module* cannot be found
  - When the parameter is omitted

[Example of use]

- To change "_sym1" in b.lib to "_data", describe as:

```
>rlink a.obj -lib=b.lib,c.lib -lib_rename=b.lib(_sym1=_data)
```

[Remark]

- If this option is specified together with the -form={object,library}, -extract or -strip option, an error will occur.
- When the -form={absolute|hexadecimal|stype|binary} option is specified, the -show=struct option cannot be specified together.
- The section name of the input library cannot be changed.
- Correct operation is not guaranteed if this option is used in combination with the compile option -Omerge_files.

-DELete

This option deletes an external symbol name or a library module.

[Specification format]

```
-DELete=suboption[, ...]
  suboption := {(symbol[, ...])|file(symbol[, ...])|module}
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes external symbol name *symbol* or library module *module*.
- Symbol names or modules in specific file *file* can be deleted.
- When a C variable name or C function name is specified, add "_" at the head of the definition name in the program.
- If there are two or more files with the same name, the priority depends on the input order.
- When a symbol is deleted using this option, the object is not deleted but the attribute is changed to the internal symbol.

[Example of use]

- To delete symbol name "_sym1" in all the files, describe as:

```
>rlink a.obj -delete=(_sym1)
```

- To delete symbol name "_sym2" in b.obj, describe as:

```
>rlink a.obj b.obj -delete=b.obj(_sym2)
```

[Remark]

- If this option is specified together with the -extract option or -strip option, this option will be invalid.
- When the -form=library option is specified, library modules can be deleted.
- When the -form={absolute|relocate|hexadecimal|stype|binary} option is specified, external symbols can be deleted.
- Operation is not guaranteed if this option is used in combination with compile option -Omerge_files.

-REPlace

This option replaces library modules.

[Specification format]

```
-REPlace=suboption[, ...]
  suboption := {file|file(module[, ...])}
```

- Interpretation when omitted
  None

[Detailed description]

- This option replaces specified file *file* or library module *module* with the module having the same name in the library file specified by the -library option.

[Example of use]

- To replace file1.obj with module "file1" in library file lib1.lib, describe as:

```
>rlink -library=lib1.lib -replace=file1.obj -form=library
```

- To replace module "mdl1" with module "mdl1" in library file lib1.lib, describe as:

```
>rlink -library=lib2.lib -replace=lib1.lib(mdl1) -form=library
```

[Remark]

- If the -form={object|relocate|absolute|hexadecimal|stype|binary} option and the -extract or -strip option is specified, this option will be invalid.
- Operation is not guaranteed if this option is used in combination with compile option -Omerge_files.

-EXTract

This option extracts library modules.

[Specification format]

```
-EXTract=module[,module]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option extracts library module *module* from the library file specified by the -library option.

[Example of use]

- To extract module "file1" from library file "lib.lib" and output it to a file with the object file output format, describe as:

```
>rlink -library=lib1.lib -extract=file1 -form=obj
```

[Remark]

- If the -form={absolute|hexadecimal|stype|binary} option and the -strip option is specified, this option will be invalid.
- When the -form=library option is specified, library modules can be deleted.
- When the -form={absolute|relocate|hexadecimal|stype|binary} option is specified, external symbols can be deleted.

-STRip

This option deletes debug information in the load module file or library file.

[Specification format]

```
-STRip
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes debug information in the load module file or library file.
- The files before debug information is deleted are backed up in file "*file-name*.abk".
- Multiple input files cannot be specified.

[Example of use]

- To delete debug information of file1.abs and output these to file1.abs, respectively, describe as:
  The files before debug information is deleted are backed up in file1.abk.

```
>rlink -strip file1.abs
```

[Remark]

- If the -form={object|relocate|hexadecimal|stype|binary} option is specified, this option will be invalid.

-CHange_message

This option changes the type of information, warning, and error messages.

[Specification format]

```
-CHange_message=suboption[, ...]
  suboption := {level|level=range[, ...]}
  range := {num|num-num}
```

- Interpretation when omitted
  None

[Detailed description]

- This option changes type *level* of information, warning, and error messages.
- The execution continuation or abort at the message output.
- The items that can be specified as *level* are shown below.

INFORMATION	Information
WARNING	Warning
ERROR	Error

- If message number *num* is specified, the type of the message with the specified number is changed.
  Also, a range of message numbers can be specified using a hyphen (-).
- Specify the 4-digit number that is output after the component number (05) and the phase of occurrence (6) as *num* (for example, specify 2310 for message number E0562310).
- If the specification of a message number is omitted, the types of all messages are changed to the specified one.

[Example of use]

- To change "E0561310" to a warning and continue the execution at the "E0561310" output, describe as:

```
>rlink a.obj b.obj -change_message=warning=1310
```

- To change all information and warning messages to error messages, describe as:
  If a message is output, the execution will abort.

```
>rlink a.obj b.obj -change_message=error
```

-Hide

This option deletes local symbol name information from the output file.

[Specification format]

```
-Hide
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes local symbol name information from the output file.
- Since the name information regarding local symbols is deleted, local symbol names cannot be checked even if the file is opened with a binary editor.
  This option does not affect the operation of the generated file.
- Use this option to keep the local symbol names secret.
- The following types of symbols are hidden.
  The entry function name is not hidden.
  - C source: Variable or function names specified with the static qualifiers
  - C source: Label names for the goto statements
  - Assembly source: Symbol names of which external definition (reference) symbols are not declared

[Example of use]

- To delete local symbol name information from the output file, describe as:

```
>rlink a.obj b.obj -hide
```

The C source example in which this option is valid is shown below.

```
int g1;
int g2=1;
const int g3=3;
static int s1;          //<--- The static variable name will be hidden.
static int s2=1;        //<--- The static variable name will be hidden.
static const int s3=2;  //<--- The static variable name will be hidden.

static int sub1()       //<--- The static variable name will be hidden.
{
        static int s1;  //<--- The static variable name will be hidden.
        int l1;

        s1 = l1; l1 = s1;
        return(l1);
}

int main()
{
        sub1();
        if (g1==1)
                goto L1;
        g2=2;
L1:                     //<--- The label name of the goto statement will be hidden.
        return(0);
}
```

[Remark]

- This option is valid only when the -form={absolute|relocate|library} option is specified.
- This option cannot be specified when a file specified by the -goptimize option at compilation or assembly is input and the relocatable or library file format is specified for the output file.
- When this option is specified with the external variable access optimization, do not specify it for the first linking, and specify it only for the second linking.
- The symbol names in the debug information are not deleted by this option.

-Total_size

This option displays the total size of sections after the linking to the standard error output.

[Specification format]

```
-Total_size
```

- Interpretation when omitted
  None

[Detailed description]

- This option displays the total size of sections after the linking to the standard error output.
- The sections are categorized as follows, with the overall size of each being displayed.
  - Executable program sections
  - Non-program sections allocated to the ROM area
  - Sections allocated to the RAM area
- This option makes it easy to see the total sizes of sections allocated to the ROM and RAM areas.

[Example of use]

- To display the total size of sections after the linking to the standard error output, describe as:

```
>rlink a.obj b.obj -total_size
```

[Remark]

- The -show=total_size option must be specified in order to output the total sizes to the link map file.
- When the -rom option has been specified for a section, that section will be used by both the origin (ROM) and destination (RAM) for the transfer of the data in the section. The sizes of such sections are to be considered in the total sizes of sections in both ROM and RAM.
- If the -form={object|relocate|library} option or -extract option is specified, this option will be invalid.

-VERBOSE [V1.10 or later]

This option displays detailed information in the standard error output.

[Specification format]

```
-VERBOSE=<sub>[, ...]
sub : CRC
```

- Interpretation when omitted
  None

[Detailed description]

- This option displays the contents specified by the suboption in the standard error output.
- The suboption below can be specified.

CRC	This suboption displays the CRC operation result and its output address. Valid when the crc option is specified.

[Example of use]

- To display the CRC operation result and its output address in the standard error output, describe as:

```
> rlink a.obj -form=stype -start=.SEC1/1000 -crc=2000=1000-10ff/CCITT -verbose=crc
```

-LOgo

This option outputs the copyright notice.

[Specification format]

```
-LOgo
```

- Interpretation when omitted
  This option outputs the copyright notice.

[Detailed description]

- This option outputs the copyright notice.

[Example of use]

- To output the copyright notice, describe as:

```
>rlink a.obj b.obj -logo
```

-NOLOgo

This option suppresses the output of the copyright notice.

[Specification format]

```
-NOLOgo
```

- Interpretation when omitted
  The copyright notice is output (It is the same result as when the -logo option is specified).

[Detailed description]

- This option suppresses the output of the copyright notice.

[Example of use]

- To suppress the output of the copyright notice, describe as:

```
>rlink a.obj b.obj -nologo
```

-END

This option executes option strings specified before this option.

[Specification format]

```
-END
```

- Interpretation when omitted
  None

[Detailed description]

- This option executes option strings specified before this option.
  After link processing is terminated, option strings specified before this option are input and link processing is continued.

[Caution]

- This option can be used only in a subcommand file.

[Example of use]

- Create subcommand file "sub.txt" with the following content.

```
input=a.obj,b.obj                          ;(1)
start=.SEC1,.SEC2,.SEC3/100,.SEC4/8000  ;(2)
output=a.abs                               ;(3)
end
input=a.abs                                ;(4)
form=stype                                 ;(5)
output=a.mot                               ;(6)
```

To specify subcommand file sub.txt, describe as:

```
>rlink -subcommand=sub.txt
```

Processing from (1) to (3) are executed and a.abs is output.
Then processing from (4) to (6) are executed and a.mot is output.

-EXIt

This option specifies the end of option specifications.

[Specification format]

```
-EXIt
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies the end of option specifications.

[Caution]

- This option can be used only in a subcommand file.

[Example of use]

- Create subcommand file "sub.txt" with the following content.

```
input=a.obj,b.obj                        ;(1)
start=.SEC1,.SEC2,.SEC3/100,.SEC4/8000  ;(2)
output=a.abs                             ;(3)
exit
```

To specify subcommand file sub.txt, describe as:

```
>rlink -subcommand=sub.txt -nodebug
```

Processing from (1) to (3) are executed and a.abs is output.
The -nodebug option specified on the command line after this option is executed is invalid.

2.5.4 Library generator options [V1.13.00 or later]

This section explains options for the library generation phase.

Table 2.10 Library Generator Options

Classification	Option	Description
Library generation control	-head	This option specifies the library to be configured.
	-lang	This option specifies the language standard of the standard library.
	-secure_malloc [Professional Edition only]	This option generates a malloc library for security facility.
Output control	-output	This option specifies the output file name.
Other	-logo / -nologo	This option controls the copyright display.
	-subcommand	This option specifies the subcommand file.

Library generation control

The library generation control options are as follows.

- -head
- -lang
- -secure_malloc [Professional Edition only]

-head

This option specifies the library to be configured.

[Specification format]

```
-head=<sub>[,...]
<sub> : { all | runtime | ctype | math | mathf | stdio | stdlib | string | inttypes }
```

- Interpretation when omitted
  It is the same result as when -head=all is specified.

[Detailed description]

- This option specifies a library to be configured with a header file name.
- If -head=all is specified, all header file names are specified as those to be configured.
- Runtime libraries are always configured.

-lang

This option specifies the language standard of the standard library.

[Specification format]

```
-lang={ c | c99 }
```

- Interpretation when omitted
  It is the same result as when -lang=c is specified.

[Detailed description]

- This option specifies the language standard of the standard library.
- When -lang=c is selected, the library includes only the components that conform to the C89 standard, and the functions expanded in the C99 standard are not included. When -lang=c99 is selected, the library is configured by components conforming to the C89 and C99 standards.
- Make sure that the specification is the same as those for the applications that reference the library.

[Remark]

- The C++ standard library is not supported.

-secure_malloc [Professional Edition only]

This option generates a malloc library for security facility.

[Specification format]

```
-secure_malloc
```

- Interpretation when omitted
  A malloc library for normal use is generated.

[Detailed description]

- This option generates a malloc library for security facility.
- When using a malloc library for security facility, the __heap_chk_fail function is called when one of the following operations is performed:
  - A pointer to an area other than that allocated by calloc, malloc, or realloc is passed to free or realloc.
  - The pointer to an area released by free is passed again to free or realloc.
  - After a value is written to an address outside the area allocated by calloc, malloc, or realloc (within two bytes before and after the allocated area), the pointer to that area is passed to free or realloc.
- The __heap_chk_fail function needs to be defined by the user. This function describes the processing to be executed when an error occurs in management of dynamic memory.
- Note the following points when defining the __heap_chk_fail function.
  - The __heap_chk_fail function should be a far function whose return value and parameter type should be the void type.
    void __far __heap_chk_fail(void);
- Do not define the __heap_chk_fail function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.
- The calloc, malloc, and realloc functions for the security facility allocate two extra bytes each before and after an allocated area for the purpose of detecting writing to addresses outside the allocated area. This consumes more heap memory area than with the usual functions.

[Caution]

- The default size of the heap memory area is 0x100 bytes.
- To change the heap memory area, define the _REL_sysheap array and set the array size in the _REL_sizeof_sysheap variable.

[Example of setting the heap memory area]

```
#include <stddef.h>
#define SIZEOF_HEAP  0x200
int _REL_sysheap[SIZEOF_HEAP / sizeof(int)];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark The _REL_sysheap array should be allocated to an even address.

Output control

The output control option is as follows.

- -output

-output

This option specifies the output file.

[Specification format]

```
-output=file
```

- Interpretation when omitted
  A standard library with the file name stdlib.lib is generated in the current folder.

[Detailed description]

- This option specifies the output file name for the standard library.
- *file* should be specified by using an absolute path or a relative path.
- A relative path used to specify *file* is interpreted as the relative path from the current folder.
- An error will occur if *file* is omitted.

Other

Other options are as follows.

- -logo / -nologo
- -subcommand

-logo / -nologo

This option controls the copyright display.

[Specification format]

```
-logo
-nologo
```

- Interpretation when omitted
  The copyright is displayed.

[Detailed description]

- When -logo is specified, the copyright is displayed. When -nologo is specified, the copyright is not displayed.
- If these options are specified at the same time, the option specified last will be valid.

-subcommand

This option specifies the subcommand file.

[Specification format]

```
-subcommand=file
```

[Detailed description]

- This option handles *file* as a subcommand file.
- An error will occur if the file specified by *file* does not exist.
- An error will occur if *file* is omitted.

Compile options that can be specified for the library generator

In addition to library options, compile options can be specified for the library generator so that they can be selected when compiling a library.

The following compile options can be specified for the library generator. If any other compile option is specified, an error occurs or the specification is ignored without a warning output.

Make sure that the specifications of these options are the same as those for the application programs that reference the library. Library-specific specifications of some options are possible. Such options are indicated in the "Individual specification" column.

Compile option name	Individual Specification	Remarks
-cpu	Not possible	
-use_mda	Not possible	
-lang	Not possible	
-dbl_size	Not possible	
-O[*level]*	Possible	
-O*item*	Possible	The following can be specified for *item*: - unroll - delete_static_func - inline_level - inline_size - pipeline - tail_call - alias - same_code - branch_chaining - align
-goptimize	Possible	
-stuff	Possible	

For details about each option, see "2.5.1 Compile options".

2.6 Specifying Multiple Options

This section describes the operation when two or more options are specified for the ccrl command at the same time.

Note the following when specifying multiple options by using -subcommand or -asmopt.

- When a file is specified through the -subcommand option, the options in the file are expanded at the location where the -subcommand option is specified on the command line. Therefore, the rules described in this section are applied to the options and location after expansion.
- When options are specified through any of -asmopt, -lnkopt, -asmcmd, and -lnkcmd, the specified options are not expanded on the command line. Therefore, the rules described in this section are not applied to these options.

2.6.1 Specifying multiple times of options

The following describes the compiler operation when the same option is specified multiple times.

Same operation as when specified only one time (with no parameter)	-V, -help, -g, -far_rom, -goptimize, -pass_source, -signed_char, -signed_bitfield, -volatile, -merge_string, -pack, -strict_std, -refs_without_declaration, -large_variable, -nest_comment, -check_language_extension, -Omerge_files, -Ointermodule, -Owhole_program, -g_line, -control_flow_integrity, -unaligned_pointer_for_ca78k0r
The parameters for all option specifications are valid	-D, -U, -I, -preinclude, -preprocess, -no_warning_num, -change_message, -subcommand, -asmopt, -asmcmd , -lnkopt, -lnkcmd
The last option specification and its location are valid, or the parameters for the last option specification are valid	-o, -obj_path, -asm_path, -prep_path, -O*level*, -Oinline_level, -Oinline_size, -Opipeline, -Ounroll, -Odelete_static_func, -Oalias, -Otail_call, -Osame_code, -switch, -character_set, -stack_protector, -stack_protector_all, -lang, -P, -S, -c, -use_mda, -memory_model, -dbl_size, -error_file, -misra2004, -misra2012, -ignore_files_misra, -misra_intermodule, -Obranch_chaining, -Oalign
Error	-cpu, -dev, -convert_cc

2.6.2 Priority of options

The following options disable other options.

-V, -h	All options will be invalid.
-P	Since execution is terminated at preprocessing, the options related to the subsequent processing after preprocessing will be invalid. Note that only the macro definitions resulted from option settings are valid even if the options themselves are invalid. Example When -P and -cpu=S1 are specified together, the operation ends after the preprocessing and no code is generated for the S1 core. However, predefined macro __RL78_S1__, which should be output when -cpu=S1 is specified, becomes valid, and the definitions depending on __RL78_S1__ (such as #ifdef definitions) are valid in the preprocessing.
-S	Since execution is terminated at compile processing, options related to the assemble processing will be invalid.
-c	Since execution is terminated at assemble processing, options related to the link processing will be invalid.
-lang=c99	-misra2004 and -convert_cc will be invalid.

Specifying any of the following options disables part of the functions of other options.

- -volatile
  The external variables and the variables specified with #pragma address are not optimized even when the -O option is specified.

- -far_rom
  The near/far attribute of ROM data is set to far regardless of whether the -memory_model option is specified.
- -Oalias
  Even when -Oalias=noansi and -strict_std are specified together, -Oalias=ansi is not valid.

If options are specified by the following combinations, the option specified last will be valid with outputting a warning.

- -P, -S, -c
- -D, -U (When their symbol names are same.)
- -Onothing, -Olite, -Odefault, -Osize, -Ospeed

Depending on the order of specified options, the following options will be invalid.

- -O*item*[Note] that is specified before -Onothing, -Olite, -Odefault, -Osize, or -Ospeed

Note -O*item* can be any of the following options.
-Ounroll, -Odelete_static_func, -Oinline_level, -Oinline_size, -Opipeline, -Otail_call, -Osame_code, -Obranch_chaining, -Oalign

2.6.3 Combinations of options with conflicting features

If options are specified by the following combinations, an error will occur.

- -dev
  If the specifications in -cpu or -use_mda do not match the contents of the device file specified by -dev, an error will occur.
- -misra2004 and –misra2012
  A compile error will occur when -misra2004 and -misra2012 are specified simultaneously.

2.6.4 Dependence between options

The behavior of the following options varies depending on what other options are specified.

-preprocess	This option will be invalid if the -P option is not specified at the same time. At this time, a warning will not be output.
-o	If the -P, -S, or -c option is specified at the same time, then the generated file types will be a preprocessed file, assembly source file, or object file.
-g	If the -O option is specified at the same time, debug information may not be output in source line units due to optimization effects.
-Oinline_level	If this option is specified at the same time with the -merge_files option, inline expansion may be performed between files.

2.6.5 Relationship with #pragma directives

The behavior of the following options varies depending on the relationship with #pragma directives.

- -cpu=S1
  If register bank specification "bank=" is used in #pragma interrupt or #pragma interrupt_brk, a compilation error will occur.
- When a #pragma-specified function or variable is declared without ___near or __far, the near/far attribute of the function or variable is affected by the settings of the -cpu, -memory_model, and -far_rom options.

2.6.6 Relationship with near and far

The near/far attribute of data and functions is determined by options and keywords.
The following shows how to determine the near/far attribute.

	Option Or Keyword	How to Determine near/far Attribute	Priority
(a)	-cpu	This option determines the default near/far attribute.	1
(b)	-memory_model	This option overwrites the default near/far attribute determined by (a).	2
(c)	-far_rom	Only for ROM data, this option overwrites the near/far attribute determined by (b) with the far attribute.	3
(d)	__near/__far	These settings are not affected by (a) to (c); the __near and _far specifications are valid.	4

For (b) and (c) in the above table (-memory_model and -far_rom options), the following shows the near/far attribute determined for ROM data and RAM data when only the former option is specified and when both options are specified.

type Value Specified in -memory_model=type	-far_rom Specification	Function	ROM Data	RAM Data
small	Not specified	near	near	near
medium		far	near	near
small	Specified	near	far	near
medium		far	far	near

3. OUTPUT FILES

This chapter explains the format and other aspects of files output by a build via each command.

3.1 Assemble List File

This section explains the assemble list file.
The assemble list is the list-formatted version of the code that is output when the source has been compiled and assembled.
It can be used to check the code resulting from compilation and assembly.

3.1.1 Structure of the assemble list

The structure and contents of the assemble list are shown below.

Output Information	Description
Assemble list information	Assembler information, location counter value, code, line number, and source program under assembly
Section list information	Type, size, and name of section
Command line information	Character string of command line of assembler

3.1.2 Assemble list information

The assembler information, location counter value, code, line number, and source program under assembly is output.
The output example of the assemble list is shown below.

```
(1)* RL78 Family Assembler VX.XX.XXx * Assemble Source List *
(2)       (3)                  (4) (5)
OFFSET    CODE                 NO  SOURCE STATEMENT

00000000                        1  #CC-RL Compiler RL78 Assembler Source
00000000                        2  #@  CC-RL Version : VX.XX.XXx [DD Mmm YYY]
00000000                        3  #@  Commmand :
00000000                        4  #@   -cpu=S3
00000000                        5  #@   -S
00000000                        6  #@   tp.c
00000000                        7  #@  compiled at Sun May 18 18:59:17 2014
00000000                        8
00000000                        9            .PUBLIC     _label
00000000                       10            .PUBLIC     _func
00000000                       11
00000000                       12            .SECTION    .textf,TEXTF
00000000                       13  _func:
00000000                       14            .STACK      _func = 4
00000000 8F0000                15            mov         a, !LOWW(_label)
00000003 D7                    16            ret
00000000                       17            .SECTION    .bss,BSS
00000000                       18            .ALIGN      2
00000000                       19  _label:
00000000                       20            .DS         (2)
```

Number	Description
(1)	Assembler information The type and version of the assembler are output.

Number	Description
(2)	Location counter value The location counter value for the beginning of the code generated for the source program of the corresponding line is output.
(3)	Code The code (machine language instruction or data) generated for the source program of the corresponding line is output. Each byte is expressed as 2-digit hexadecimal number. Example When "8F0000" is output in the list, "8F", "00", and "00" are stored from the lower bytes.
(4)	Line number The number of the line is output. The lines where include files are expanded are also counted. This is expressed in a decimal number.
(5)	Source program The source program of the line is output. Compiler information (lines 1 to 4) is output only when an assembly source file output from the compiler is assembled.

Remark The output of instructions DIVHU and DIVWU in an assemble list.

In the assemble list, the DIVHU and DIVWU instructions are shown as follows.
The DIVHU and DIVWU instructions in the assembly source program are each expanded by macro expansion into the DIVHU and NOP instructions and the DIVWU and NOP instructions.

Example
Input program example including DIVHU and DIVWU (sample.asm).

```
DIVHU   ; comment1
DIVWU   ; comment2
```

Output program example including DIVHU and DIVWU (a part of sample.prn).

```
00000000                1  DIVHU       ; comment1
00000000 CEFB03         2  -- div**
00000003 00             3  -- nop
00000004                4  DIVWU       ; comment2
00000004 CEFB0B         5  -- div**
00000007 00             6  -- nop
```

3.1.3 Section list information

The type, size, and name of the section is output.
The output example of the section list is shown below.

```
Section List
(1)     (2)                 (3)
Attr    Size                Name

TEXTF   4 (00000004)        .textf
BSS     2 (00000002)        .bss
```

Number	Description
(1)	Section type The relocation attribute of the section is output.
(2)	Section size The size of the section is output. This is expressed in a decimal number and also expressed in hexadecimal number in parentheses.
(3)	Section name The name of the section is output.

3.1.4 Command line information

The character string of the command line of the assembler is output.
The output example of the command line information is shown below.

```
Command Line Parameter
-cpu=S3 tp.asm -prn_path        (1)
```

Number	Description
(1)	Character string of command line The character string of the command line specified for the assembler is output.

3.2 Link Map File

This section explains the link map file.
The link map has information of the link result. It can be referenced for information such as the section's allocation addresses.

3.2.1 Structure of link map

The structure and contents of the link map are shown below.

Output Information	Description	-show Option Specification	When -show Option Is Omitted
Header information	Version information of the optimizing linker and time of linkage	-	Output
Option information	Option strings specified by a command line or subcommand file	-	Output
Error information	Error message	-	Output
Link map information	Section name, start/end addresses, size, and type	-	Output
	When -show=relocation_attribute is specified, the relocation attribute is output.	-show=relocation_attribute	No output
Total section size	Total sizes of RAM, ROM, and program sections	-show=total_size	No output
Symbol information	Static defined symbol name, address, size, type (in the order of address), and whether optimization is applied When the -show=reference is specified, the reference count of each symbol is also output. When -show=struct is specified, information on the structure and union members is output.	-show=symbol -show=reference -show=struct	No output
Contents of the function list	Contents of the function list for use in detecting illegal indirect function calls	-show=cfi	No output
Cross reference information	Symbol reference information	-show=xreference	No output
Vector table address information	Contents of the vector table addresses	-show=vector	No output
CRC information	CRC operation result and its output address	-	Always output when the -crc option is specified

Caution The -show option is valid when the -list option is specified.
See "-SHow" for details about the -show option.

3.2.2 Header information

The version information of the optimizing linker and the time of linkage are output.
The output example of the header information is shown below.

```
Renesas Optimizing Linker (VX.XX.XX)            XX-Xxx-XXXX XX:XX:XX    (1)
```

Number	Description
(1)	Version information of the optimizing linker and time of linkage The version information of the optimizing linker and the time of linkage are output.

3.2.3 Option information

Option strings specified by a command line or subcommand file are output.
The output example of the option information when the following command line and subcommand file are specified is shown below.

<Command line>

```
>rlink -subcommand=sub.txt -list -show
```

<Subcommand file "sub.txt">

```
input sample.obj
```

```
*** Options ***

-subcommand=sub.txt     (1)
input sample.obj        (2)
-list                   (1)
-show                   (1)
```

Number	Description
(1)	Options specified by command line The options specified by the command line are output (in their specified order).
(2)	Options specified in subcommand file The options specified in subcommand file "sub.txt" are output.

3.2.4 Error information

Error messages are output.
The output example of the error information is shown below.

```
*** Error Information ***

** E0562310:Undefined external symbol "_func_02" referenced in "sample.obj"     (1)
```

Number	Description
(1)	Error message Error messages are output.

3.2.5 Link map information

Start/end addresses, size, and type of each section are output in the order of address.
The output example of the link map information is shown below.

```
*** Mapping List ***

(1)                                (2)       (3)         (4)     (5)
SECTION                            START     END         SIZE    ALIGN

.textf
                                   00000100  0000013b      3c    1
.data
                                   000f0400  000f0403       4    2
.bss
                                   000f0404  000f040b       8    2
```

Number	Description
(1)	Section name The name of the section is output.
(2)	Start address The start address is output. This is expressed in a hexadecimal number.
(3)	End address The end address is output. This is expressed in a hexadecimal number.
(4)	Section size The section size is output (byte). This is expressed in a hexadecimal number.
(5)	Section alignment size The section alignment size is output.

When -show=relocation_attribute is specified, the relocation attribute corresponding to the section is output. An output example of the relocation attribute is shown below.

```
*** Mapping List ***

SECTION                            START     END         SIZE    ALIGN   ATTRIBUTE
                                                                         (1)

.textf
                                   00000100  0000013b      3c    1       TEXTF
.data
                                   000f0400  000f0403       4    2       DATA
.bss
                                   000f0404  000f040b       8    2       BSS
```

Number	Description	
(1)	Relocation attribute The relocation attribute of the section is output. It is output as shown below in response to the code written in the assembly language.	
	Relocation attribute	Link map information
	CALLT0	CALLT0
	TEXT	TEXT
	TEXTF	TEXTF
	TEXTF_UNIT64KP	TEXTF_UNIT64KP
	AT	TEXT_AT
	CONST	CONST
	CONSTF	CONSTF
	DATA	DATA
	DATAF	DATAF
	SDATA	SDATA
	DATA_AT	DATA_AT
	BSS	BSS
	BSSF	BSSF
	SBSS	SBSS
	BSS_AT	BSS_AT
	OPT_BYTE	OPT_BYTE
	SECUR_ID	SECUR_ID
	FLASH_SECUR_ID	FLASH_SECUR_ID
	OTHER	OTHER

3.2.6 Total section size

When the -show=total_size option is specified, the total sizes of RAM, ROM, and program sections are output.
The output example of the total section size is shown below.

```
*** Total Section Size ***

RAMDATA SECTION:        00000660 Byte(s) (1)
ROMDATA SECTION:        00000174 Byte(s) (2)
PROGRAM SECTION:        000016d6 Byte(s) (3)
```

Number	Description
(1)	Total size of RAM data sections The total size of RAM data sections is output. This is expressed in a hexadecimal number.
(2)	Total size of ROM data sections The total size of ROM data sections is output. This is expressed in a hexadecimal number.
(3)	Total size of program sections The total size of program sections is output. This is expressed in a hexadecimal number.

3.2.7 Symbol information

When the -show=symbol option is specified, the external defined symbol or static internal defined symbol address, size, type, and whether optimization is applied are output in the order of address.
When the -show=reference option is specified, the reference count of each symbol is also output.
The output example of the symbol information is shown below.

```
*** Symbol List ***

SECTION=(1)
FILE=(2)
                (3)          (4)              (5)
                START        END              SIZE
  (6)           (7)          (8)              (9)           (10)    (11)
  SYMBOL        ADDR         SIZE             INFO          COUNTS  OPT

SECTION=.text
FILE=sample.obj
                00000100     00000123         24
  _main
                00000100            0         func ,g             0
  _func_01
                00000118            0         func ,g             0
SECTION=.bss
FILE=sample.obj
                000f0404     000f040b         8
  _gvall
                000f0404            4         data ,g             0
```

Number	Description
(1)	Section name The name of the section is output.
(2)	File name The file name is output.
(3)	Start address The start address of the corresponding section included in the file shown in (2) is output. This is expressed in a hexadecimal number.
(4)	End address The end address of the corresponding section included in the file shown in (2) is output. This is expressed in a hexadecimal number.
(5)	Section size The size of the corresponding section included in the file shown in (2) is output (in byte units). This is expressed in a hexadecimal number.
(6)	Symbol name The symbol name is output.
(7)	Symbol address The symbol address is output. This is expressed in a hexadecimal number.
(8)	Symbol size The symbol size is output (in byte units). This is expressed in a hexadecimal number.
(9)	Symbol type The data type and declaration type are output. - Data type func: Function name data: Variable name entry: Entry function name none: Undefined (label, assembler symbol) - Declaration type g: External definition l: Internal definition

Number	Description
(10)	Reference count of symbol The reference count of the symbol is output. This is expressed in a hexadecimal number. This item is output only when the -show=reference option is specified. When the reference count of the symbol is not output, "*" is output.
(11)	Whether optimization is applied Whether optimization is applied is output. ch: Symbol changed by optimization cr: Symbol generated by optimization mv: Symbol moved by optimization

When the -show=struct option is specified, the addresses for the structure and union members that are defined in the source file for which the -g option was specified at compilation are output.

The output example of the symbol information is shown below.

```
*** Symbol List ***

SECTION
FILE
                START       END             SIZE
  SYMBOL        ADDR        SIZE            INFO          COUNTS  OPT
    (1)                     (2)
  STRUCT                    SIZE
    (3)         (4)         (5)             (6)
  MEMBER        ADDR        SIZE            INFO

SECTION=B
FILE=sample.obj
                00001000    00001003        4
  a
                00001000    4               data ,g       1
  struct A{
                            4
  a.b
                00001000    1               char
  a.c
                00001002    2               short
```

Number	Description
(1)	Type name The type name of the structure or union is output.
(2)	Size The size of the structure or union is output.
(3)	Name of member The names of the members of the structure or union are output.
(4)	Address of member The addresses of the members of the structure or union are output.
(5)	Size of member The sizes of the members of the structure or union are output. For a bit field, the type size of the member of the structure or union is output.

Number	Description
(6)	Type name of member The type names of the members of the structure or union are output. For a bit field, the type name of the member of the structure or union is output. For the pointer type, the following is output. [pointer]: When __near or __far is not specified [near pointer]: When the pointer is specified to be a near pointer [far pointer]: When the pointer is specified to be a far pointer

3.2.8 Contents of the function list

If show=cfi is specified, this option outputs the contents of the function list for use in detecting illegal indirect function calls.

The output example is given below.

```
*** CFI Table List ***

SYMBOL/ADDRESS

_func     (1)
0000F100  (2)
```

Number	Description
(1)	Outputs the symbol for the function.
(2)	Outputs the address of the function if a symbol for it has not been defined.

3.2.9 Cross reference information

When the -show=xreference option is specified, the reference information of symbols (cross reference information) is output.

The output example of the cross reference information is shown below.

```
*** Cross Reference List ***

(1)  (2)        (3)            (4)       (5)
No   Unit Name  Global.Symbol  Location  External Information
0001 sample1
     SECTION=.text
                _main
                               00000100
                _func_01
                               00000118
     SECTION=.data
                _gval3
                               000f0400  0003(00000032:.text)
                                         0003(00000038:.text)
     SECTION=.bss
                _gval1
                               000f0404  0001(0000001a:.text)
                                         0001(00000020:.text)
                _gval2
                               000f0408  0002(00000026:.text)
                                         0002(0000002c:.text)
0002 sample2
     SECTION=.text
                _func02
                               00000124  0001(0000000a:.text)
0003 sample3
     SECTION=.text
                _func03
                               00000130  0001(00000010:.text)
```

Number	Description
(1)	Unit number The identification number in object units is output.
(2)	Object name The object name is output in the order of input when linking.
(3)	Symbol name The symbol name is output in the ascending order of allocation address for each section.
(4)	Symbol allocation address The symbol allocation address is output. When the -form=relocate option is specified, this is a relative value from the start of the section.
(5)	Address of external symbol that has been referenced The address of the external symbol that has been referenced is output. *Unit number* (*address or offset in section:section name*)

3.2.10 Vector table address information

When the -show=vector option is specified, the contents of the vector table addresses is output.
The output example of the vector table address information is shown below.

```
*** Variable Vector Table List ***
(1)     (2)
ADDRESS SYMBOL/ADDRESS
00      start
02      dummy
04      INTWDTI
06      0000F100
 :
```

Number	Description
(1)	Vector table address The vector table address is output.
(2)	Symbol The symbol is output. When no symbol is specified, the address is output.

3.2.11 CRC information

When the -crc option is specified, the CRC operation result and its output address is output.
The output example of the CRC information is shown below.

```
***CRC Code***
CODE:    cbob       (1)
ADDRESS: 00007ffe   (2)
```

Number	Description
(1)	CRC operation result The CRC operation result is output.
(2)	Address of CRC operation result output The address of CRC operation result output is output.

3.3 Link Map File (When Objects Are Combined)

This section explains the contents and format of the link map file that is output by the optimizing linker when the input file is an Intel HEX file or a Motorola S-record file.

3.3.1 Structure of link map

The structure and contents of the link map are shown below.

Output Information	Description
Header information	Version information of the optimizing linker and time of linkage
Option information	Option strings specified by a command line or subcommand file
Error information	Error message
Entry information	Execution start address
Combined address information	Combined source files, and start and end addresses and size of continuous range data
Address overlap information	Overlapped combine source files, and start and end addresses and size of overlapped range data

3.3.2 Header information

The version information of the optimizing linker and the time of linkage are output.
The output example of the header information is shown below.

```
Renesas Optimizing Linker (VX.XX.XX)            XX-Xxx-XXXX XX:XX:XX    (1)
```

Number	Description
(1)	Version information of the optimizing linker and time of linkage The version information of the optimizing linker and the time of linkage are output.

3.3.3 Option information

Option strings specified by a command line or subcommand file are output.
The output example of the option information when the following command line and subcommand file are specified is shown below.

<Command line>

```
>rlink -subcommand=sub.txt -list
```

<Subcommand file "sub.txt">

```
input sample1.mot
input sample2.mot
form stype
output result
```

```
*** Options ***

-subcommand=sub.txt     (1)
input sample1.mot       (2)
input sample2.mot       (2)
form stype              (2)
output result           (2)
-list                   (1)
```

Number	Description
(1)	Options specified by command line The options specified by the command line are output (in their specified order).
(2)	Options specified in subcommand file The options specified in subcommand file "sub.txt" are output.

3.3.4 Error information

Error messages are output.
The output example of the error information is shown below.

```
*** Error Information ***

E0562420:"sample1.mot" overlap address "sample2.mot" : "00000100"       (1)
```

Number	Description
(1)	Error message Error messages are output.

3.3.5 Entry information

The execution start address is output.
The output example of the entry information is shown below.

```
*** Entry address ***
00000100        (1)
```

Number	Description
(1)	Execution start address The execution start address is output. However, if the execution start address is "00000000", it is not output.

3.3.6 Combined address information

The combined source files, and the start and end addresses and size of the continuous range data are output.
The output example of the combined address information is shown below.

```
*** Combine information ***
(1)             (2)         (3)         (4)
FILE            START       END         SIZE
sample1.mot
                00000100    00000127    28
sample1.mot
                00000200    00000227    28
sample2.mot
                00000250    00000263    14
sample2.mot
                00000300    0000033b    3c
```

Number	Description
(1)	Combined source file name The combined source file name is output.
(2)	Start addresses of continuous range data The start addresses of the continuous range data are output. This is expressed in a hexadecimal number.
(3)	End addresses of continuous range data The end addresses of the continuous range data are output. This is expressed in a hexadecimal number.
(4)	Size of continuous range data The size of the continuous range data is output (in byte units). This is expressed in a hexadecimal number.

3.3.7 Address overlap information

The overlapped combine source files, and the start and end addresses and size of the continuous range data are output.

The output example of the address overlap information is shown below.

```
*** Conflict information ***
(1)             (2)         (3)         (4)
FILE            START       END         SIZE
Conflict 1
                00000200    00000213    14
sample1.mot
sample2.mot
```

Number	Description
(1)	Overlapped combine source file name The overlapped combine source file name is output.
(2)	Start addresses of overlapped range data The start addresses of the overlapped range data are output. This is expressed in a hexadecimal number.
(3)	End addresses of overlapped range data The end addresses of the overlapped range data are output. This is expressed in a hexadecimal number.
(4)	Size of overlapped range data The size of the overlapped range data is output (in byte units). This is expressed in a hexadecimal number.

3.4 Library List File

This section explains the library list file.
The library list has information from the library creation result.

3.4.1 Structure of the library list

The structure and contents of the library list are shown below.

Output Information	Description	-show Option Specification	When -show Option Is Omitted
Option information	Option strings specified by a command line or subcommand file	-	Output
Error information	Error message	-	Output
Library information	Library information	-	Output
Module, section, and symbol information within the library	Module within the library	-	Output
	Symbol names within a module	-show=symbol	No output
	Section names and symbol names within each module	-show=section	No output

Caution The -show option is valid when the -list option is specified.
See "-SHow" for details about the -show option.

3.4.2 Option information

Option strings specified by a command line or subcommand file are output.
The output example of the option information when they are specified by a command line or subcommand file as follows is shown below.

<Command line>

```
>rlink -subcommand=sub.txt -list -show
```

<Subcommand file "sub.txt">

```
form library
input extmod1
input extmod2
output usrlib.lib
```

```
*** Options ***

-subcommand=sub.txt     (1)
form library            (2)
input extmod1           (2)
input extmod2           (2)
output usrlib.lib       (2)
-list                   (1)
-show                   (1)
```

Number	Description
(1)	Options specified by command line The options specified by the command line are output (in their specified order).

Number	Description
(2)	Options specified in subcommand file The options specified in subcommand file "sub.txt" are output.

3.4.3 Error information

Messages for errors or warnings are output.
The output example of the error information is shown below.

```
*** Error Information ***

** E0561200:Backed up file "sample1.lib" into "usrlib.lbk"      (1)
```

Number	Description
(1)	Message The message is output.

3.4.4 Library information

The type of the library is output.
The output example of the library information is shown below.

```
*** Library Information ***

LIBRARY NAME=usrlib.lib (1)
CPU=RL78                (2)
ENDIAN=Little           (3)
ATTRIBUTE=user          (4)
NUMBER OF MODULE=2      (5)
```

Number	Description
(1)	Library name The library name is output.
(2)	Microcontroller name The microcontroller name is output.
(3)	Endian type The endian type is output.
(4)	Library file attribute Either a system library or user library is output.
(5)	Number of modules within library The number of modules within the library is output.

3.4.5 Module, section, and symbol information within the library

Modules within the library is output.
When the -show=symbol option is specified, symbol names within the module is output.
When the -show=section option is specified, section names within the module is also output.
The output example of the module, section, and symbol information within the library is shown below.

```
*** Library List ***

(1)             (2)
MODULE          LAST UPDATE
  (3)
  SECTION
    (4)
    SYMBOL
extmod1
                12-Dec-2011 16:30:00
  .text
    _func_01
    _func_02
extmod2
                12-Dec-2011 16:30:10
  .text
    _func_03
    _func_04
```

Number	Description
(1)	Module name The module name is output.
(2)	Module definition date The module definition date is output. If the module is updated, the date of the latest update is output.
(3)	Name of section within module The name of the section within the module is output.
(4)	Name of symbol within section The name of the symbol within the section is output.

3.5 Intel HEX File

This section explains the Intel HEX file.

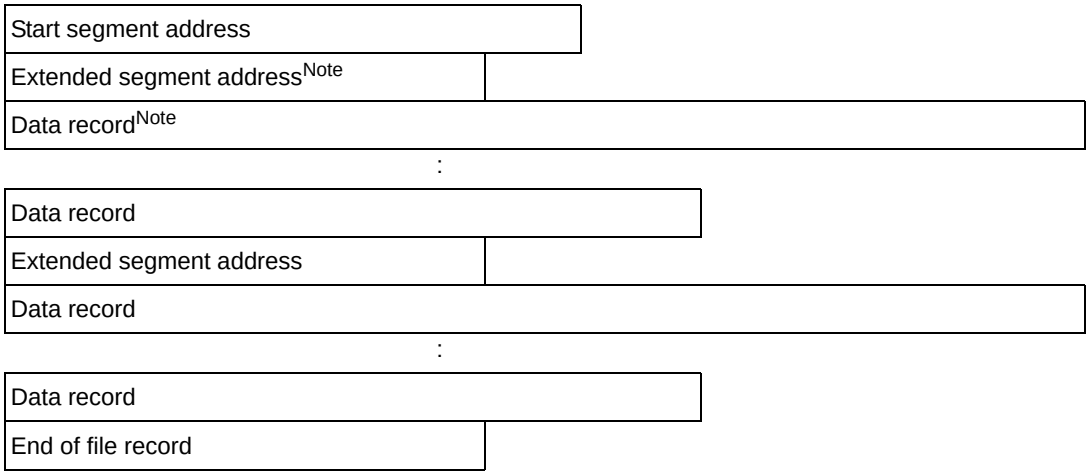
3.5.1 Structure of the Intel HEX file

The Intel HEX file (20 bits) consists of four records[Note]: start segment address record, extended segment address record, data record, and end record.
The Intel HEX file (32 bits) consists of six records[Note]: start linear address record, extended linear address record, start segment address record, extended segment address record, data record, and end of file record.

Note Each record is output in ASCII code.

The structure and contents of the Intel HEX file are shown below.

Figure 3.1 Structure of Intel HEX File



Note The extended segment address and data record are repeated.

Output Information	Description
Start linear address record	Linear address
Extended linear address record	Upper 16-bit address at bits 32 to 16
Start segment address record	Entry point address
Extended segment address record	Paragraph value of load address
Data record	Value of code
End of file record	End of code

Each record consists of the following fields.

```
:   XX  XXXX  XX  DD......DD SS  NL
(1) (2) (3)   (4)     (5)    (6) (7)
```

Number	Description
(1)	Record mark
(2)	Number of bytes The number of bytes is expressed as 2-digit hexadecimal number of (5).
(3)	Location address

Number	Description
(4)	Record type 05: Start linear address record 04: Extended linear address record 03: Start segment address 02: Extended segment address 00: Data record 01: End of file record
(5)	Code Each byte of code is expressed as 2-digit hexadecimal number.
(6)	Checksum This is the 2-digit two's complement value of a result of hexadecimal addition of all bytes in the record except for ":", "SS", and "NL".
(7)	Newline (\n)

Remark The location address in the Intel HEX format is 2 bytes (16 bits).
Therefore, only a 64-Kbyte space can be directly specified.
To extend this area, the Intel HEX format adds the 16-bit extended address so that a space of up to 1 Mbyte (20 bits) can be used.
Specifically, the record type that specifies the 16-bit extended address is added.
This extended address is shifted by four bits and added to the location address to express a 20-bit address.

3.5.2 Start linear address record

This indicates the linear address.

```
:   04  0000  05  XXXXXXXX  SS  NL
(1) (2) (3)   (4)    (5)    (6) (7)
```

Number	Description
(1)	Record mark
(2)	Fixed at 04
(3)	Fixed at 0000
(4)	Record type (Fixed at 05)
(5)	Linear address value
(6)	Checksum
(7)	Newline

3.5.3 Extended linear address record

This indicates the upper 16-bit address at bits 32 to 16.

```
:   02  0000  04  XXXX  SS  NL
(1) (2) (3)   (4)  (5)  (6) (7)
```

Number	Description
(1)	Record mark
(2)	Fixed at 02

Number	Description
(3)	Fixed at 0000
(4)	Record type (Fixed at 04)
(5)	Upper 16-bit address at bits 32 to 16
(6)	Checksum
(7)	Newline

Note The location address of the data record is used as the lower 16 bits.

3.5.4 Start segment address record

This indicates the entry point address.

```
:   04  0000  03  PPPP  XXXX  SS  NL
(1) (2) (3)   (4)  (5)   (6)  (7) (8)
```

Number	Description
(1)	Record mark
(2)	Fixed at 04
(3)	Fixed at 0000
(4)	Record type (Fixed at 03)
(5)	Paragraph value of entry point address[Note]
(6)	Offset value of entry point address
(7)	Checksum
(8)	Newline

Note The address is calculated by (paragraph value << 4) + offset value.

3.5.5 Extended segment address record

This indicates the paragraph value of the load address[Note].

Note This is output at the beginning of the segment (when the data record is output) or when the offset value of the data record's load address exceeds the maximum value of 0xffff and a new segment is output.

```
:   02  0000  02  PPPP  SS  NL
(1) (2) (3)   (4)  (5)  (6) (7)
```

Number	Description
(1)	Record mark
(2)	Fixed at 02
(3)	Fixed at 0000
(4)	Record type (Fixed at 02)
(5)	Paragraph value of segment
(6)	Checksum

Number	Description
(7)	Newline

3.5.6 Data record

This indicates the value of the code.

```
:   XX  XXXX  00  DD......DD  SS  NL
(1) (2) (3)   (4)     (5)     (6) (7)
```

Number	Description
(1)	Record mark
(2)	Number of bytes[Note]
(3)	Location address
(4)	Record type (Fixed at 00)
(5)	Code Each byte of code is expressed as 2-digit hexadecimal number.
(6)	Checksum
(7)	Newline

Note This is limited to the range of 0x1 to 0xff (the minimum value for the number of bytes of the code indicated by one data record is 1 and the maximum value is 255).

Example

```
:   04  0100  00  3C58E01B  6C  NL
(1) (2) (3)   (4)    (5)    (6) (7)
```

Number	Description
(1)	Record mark
(2)	Number of bytes of 3C58E01B expressed as 2-digit hexadecimal numbers
(3)	Location address
(4)	Record type 00
(5)	Each byte of code is expressed as 2-digit hexadecimal number.
(6)	Checksum The lower 1 byte of E6C, which is the two's complement of 04 + 01 + 00 + 00 + 3C + 58 + E0 + 1B = 194, is expressed as a 2-digit hexadecimal number.
(7)	Newline (\n)

3.5.7 End of file record

This indicates the end of the code.

```
:   00  0000  01  FF  NL
(1) (2) (3)   (4) (5) (6)
```

Number	Description
(1)	Record mark
(2)	Fixed at 00
(3)	Fixed at 0000
(4)	Record type (Fixed at 01)
(5)	Fixed at FF
(6)	Newline

3.6 Motorola S-record File

This section explains the Motorola S-record file.

3.6.1 Structure of the Motorola S-record file

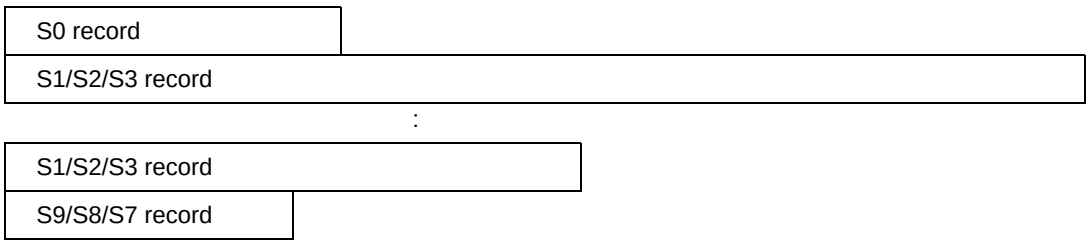
The Motorola S-record file consists of seven records[Note 1]: S0 record as the header record, S1, S2, and S3 records as the data record, and S9, S8, and S7 records as the termination records[Note 2].

Note 1. Each record is output in ASCII code.

Note 2. The Motorola S-record files are divided into three types: 16-bit address type, (24-bit) standard address type, and 32-bit address type. The format of the 16-bit address type consists of S0, S1, and S9 records, the format of the standard address type consists of S0, S2, and S8 records, and the format of the 32-bit address type consists of S0, S3, and S7 records.

The structure and contents of the Motorola S-record file are shown below.

Figure 3.2 Structure of Motorola S-record File



Output Information	Description
S0 record	File name
S1 record	Value of code
S2 record	Value of code
S3 record	Value of code
S7 record	Entry point address
S8 record	Entry point address
S9 record	Entry point address

Each record consists of the following fields.

```
Sx  XX  YY......YY  SS  NL
(1) (2)    (3)     (4) (5)
```

Number	Description
(1)	Record type S0: S0 record S1: S1 record S2: S2 record S3: S3 record S4: S4 record S5: S5 record S6: S6 record S7: S7 record S8: S8 record S9: S9 record

Number	Description
(2)	Record length The number of bytes as 2-digit hexadecimal number of (3) + number of bytes expressed by "SS"[Note].
(3)	Field
(4)	Checksum The one's complement is obtained from the sum of the number of 2-digit hexadecimal bytes in the record except for Sx, SS, and NL, and the lower one byte of the one's complement is expressed as a 2-digit hexadecimal number.
(5)	Newline (\n)

Note This is 1 byte.

3.6.2 S0 record

This indicates the file name.

```
S0  0E  0000  XX......XX  SS  NL
(1) (2)  (3)      (4)     (5) (6)
```

Number	Description
(1)	Fixed at S0
(2)	Fixed at 0E
(3)	Fixed at 0000
(4)	File name (eight characters) + file format (three characters) in most cases
(5)	Checksum
(6)	Newline

3.6.3 S1 record

This indicates the value of the code.

```
S1  XX  YYYY  ZZ......ZZ  SS  NL
(1) (2)  (3)      (4)     (5) (6)
```

Number	Description
(1)	Fixed at S1
(2)	Record length
(3)	Load address 16 bits (0x0 to 0xFFFF)
(4)	Code Each byte of code is expressed as 2-digit hexadecimal number.
(5)	Checksum
(6)	Newline

3.6.4 S2 record

This indicates the value of the code.

```
S2  XX  YYYYYY  ZZ......ZZ  SS  NL
(1) (2)  (3)       (4)     (5) (6)
```

Number	Description
(1)	Fixed at S2
(2)	Record length
(3)	Load address 24 bits (0x0 to 0xFFFFFF)
(4)	Code Each byte of code is expressed as 2-digit hexadecimal number.
(5)	Checksum
(6)	Newline

3.6.5 S3 record

This indicates the value of the code.

```
S3  XX  YYYYYYYY  ZZ......ZZ  SS  NL
(1) (2)  (3)         (4)     (5) (6)
```

Number	Description
(1)	Fixed at S3
(2)	Record length
(3)	Load address 32 bits (0x0 to 0xFFFFFFFF)
(4)	Code Each byte of code is expressed as 2-digit hexadecimal number.
(5)	Checksum
(6)	Newline

3.6.6 S7 record

This indicates the entry point address.

```
S7  XX  YYYYYYYY  SS  NL
(1) (2)  (3)     (4) (5)
```

Number	Description
(1)	Fixed at S7
(2)	Record length
(3)	Entry point address 32 bits (0x0 to 0xFFFFFFFF)
(4)	Checksum

Number	Description
(5)	Newline

3.6.7 S8 record

This indicates the entry point address.

```
S8  XX  YYYYYY  SS  NL
(1) (2)   (3)   (4) (5)
```

Number	Description
(1)	Fixed at S8
(2)	Record length
(3)	Entry point address 24 bits (0x0 to 0xFFFFFF)
(4)	Checksum
(5)	Newline

3.6.8 S9 record

This indicates the entry point address.

```
S9  XX  YYYY  SS  NL
(1) (2)  (3)  (4) (5)
```

Number	Description
(1)	Fixed at S9
(2)	Record length
(3)	Entry point address 16 bits (0x0 to 0xFFFF)
(4)	Checksum
(5)	Newline

3.7 Variable/Function Information File

This section explains the variable/function information file.
The variable/function information file is in the text format and contains declarations of the saddr variable or callt function for variables and functions defined in the C source file.

3.7.1 Outputting the variable/function information file

- When the -lnkopt=-vfinfo option is specified, the optimizing linker outputs the variable/function information file.
  For details on the -vfinfo option, see "-VFINFO".
- The optimizing linker significantly reduces the size of the code as a whole by selecting variables and functions on the basis of size of variables and frequency of reference and producing header files (variable/function information files) in which #pragma directives to use the saddr variables, callt function, or near function are added.
  For #pragma saddr, see "Using saddr area (__saddr)". For #pragma callt, see "callt function (__callt)". For #pragma near, see "near/far function (#pragma near/#pragma far) [V1.05 or later]".
- The following variables or functions are not targets of the variable/function information file.
  However, interrupt handlers are commented out in the variable/function information file.
  - Standard library functions and runtime library functions
  - Software interrupt handler, hardware interrupt handler, and RTOS interrupt handler
  - Variables or functions defined in the assembly source file
- The output example of the variable/function information file is shown below.

```
/* RENESAS OPTIMIZING LINKER GENERATED FILE 2014.10.20 */
/*** variable information ***/
(1)                  (2)      (3)    (4)   (5)
#pragma saddr var1 /* count:10,size:1,near, file1.obj */
(6)                         (2)      (3)    (4)  (7)    (5)
/* #pragma saddr var2 */ /* count: 0,size:2,near,unref, file2.obj */

/*** function information ***/
(8)                   (9)      (10)  (11)
#pragma callt func1 /* count:20,near, file1.obj */
(12)                  (9)      (10) (11)
#pragma near func2 /* count:10,far, file2.obj */
(13)                        (9)      (10) (14)  (11)
/* #pragma near func3 */ /* count: 0,far,unref, file3.obj */
```

Number	Description
(1)	Variable information A declaration of the saddr variable by a #pragma directive is output.
(2)	Number of references The number of times the variable is referenced is output.
(3)	Size of variable The size of the variable is output.
(4)	Reference type The original reference type of the variable is output as near, far, or saddr.
(5)	File name The object file name to which the variable belongs is output.
(6)	Variable information The variable spilled from the saddr area is output as a comment.

Number	Description
(7)	Supplementary information of variable Supplementary information of the variable is output. The following items are such supplementary information. unref: Output when the variable is not referenced. const: Output for a const variable. fix: Output when the location of the variable is not relocatable. unrecognizable: Output when the variable cannot be recognized as a symbol.
(8)	Function information A declaration of the callt function by a #pragma directive is output.
(9)	Number of references The number of times the function is referenced is output.
(10)	Method of function call The original method of function call is output as near, far, or callt.
(11)	File name The object file name to which the function belongs is output.
(12)	Function information A declaration of the near function by a #pragma directive is output.
(13)	Function information The function spilled from the callt or near area is output as a comment.
(14)	Supplementary information of function Supplementary information of the function is output. The following items are such supplementary information. unref: Output when the function is not referenced. interrupt: Output for an interrupt handler. unrecognizable: Output when the function cannot be recognized as a symbol.

3.7.2 How to use variable/function information file

- Add #include directive to the C source files to include variable/function information file, and then compile the files. That will reduce the size of object codes.
- Specifying -preinclude option is another way to include the information file.
  In this case, no modification is required for C source files.
- Variable/function information file can be edited manually.
  This enables users to tune the code size by enabling/disabling saddr specification for global variables or callt/near specification for functions in the variable/function information file and by adding/removing saddr specification for static variables or callt/near specification for static functions in the C source files.

4. COMPILER LANGUAGE SPECIFICATIONS

This chapter explains Compiler language specifications (basic language specification, extended language specifications, etc.)supported by the CC-RL.

4.1 Basic Language Specifications

This section explains the implementation-defined behavior of the CC-RL which is compliant with the C90 and C99 standards.

See "4.2 Extended Language Specifications" for extended language specifications explicitly added by the CC-RL.

4.1.1 Implementation-defined behavior of C90

This section covers the implementation-defined behavior given by the C90 standard.

(1) How to identify diagnostic messages (5.1.1.3).
Refer to "10. MESSAGE".

(2) The semantics of the arguments to main (5.1.2.2.1).
Not defined because of a freestanding environment.

(3) What constitutes an interactive device (5.1.2.3).
Not defined for the configuration of an interactive device.

(4) The number of significant initial characters (beyond 31) in an identifier without external linkage (6.1.2).
The entire identifier is handled as meaningful. The length of an identifier is unlimited.

(5) The number of significant initial characters (beyond 6) in an identifier with external linkage (6.1.2).
The entire identifier is handled as meaningful. The length of an identifier is unlimited.

(6) Whether case distinctions are significant in an identifier with external linkage (6.1.2).
Uppercase and lowercase characters are distinguished in identifiers.

(7) The members of the source and execution character sets, except as explicitly specified in the Standard (5.2.1).
The values of elements of the source code and execution character set are ASCII codes, EUC, SJIS, UTF-8, big5, and gb2312.
Japanese and Chinese characters are supported in comments and character strings.

(8) The shift states used for the encoding of multibyte characters (5.2.1.2).
No shift state is supported.

(9) The number of bits in a character in the execution character set (5.2.4.2.1).
8 bits.

(10) The mapping of members of the source character set (in character constants and string literals) to members of the execution character set (6.1.3.4).
Associated with the element having the same value.

(11) The value of an integer character constant that contains a character or escape sequence not represented in the basic execution character set or the extended character set for a wide character constant (6.1.3.4).
Specific non-graphical characters can be expressed by means of extended notation, consisting of a backslash (\) followed by a lower-case letter. The following are available: \a, \b, \f, \n, \r, \t, and \v. There is no other extended notation; other letters following a backslash (\) become that letter.

Escape Sequence	Value (ASCII)
\a	0x07
\b	0x08
\f	0x0C
\n	0x0A
\r	0x0D
\t	0x09

Escape Sequence	Value (ASCII)
\v	0x0B

(12) The value of an integer character constant that contains more than one character or a wide character constant that contains more than one multibyte character (6.1.3.4).
A simple character constant consisting of up to two characters has a two-byte value with the lower byte being the last character and the upper byte being the start character. A character constant having three or more characters results in an error. A character which is not represented by basic execution environment character set is regarded as a simple character constant having that value. In an invalid escape sequence, the backslash is ignored and the next character is regarded as a simple character constant.

(13) The current locale used to convert multibyte characters into corresponding wide characters (codes) for a wide character constant (6.1.3.4).
Locale is not supported.

(14) Whether a "plain" char has the same range of values as signed char or unsigned char (6.2.1.1).
The char type has the same range of values, the same representation format and the same behavior as the unsigned char type. However, it can be switched to the signed char type by option -signed_char.

(15) The representations and sets of values of the various types of integers (6.1.2.5).
Refer to "4.1.3 Internal representation and value area of data".

(16) The result of converting an integer to a shorter signed integer, or the result of converting an unsigned integer to a signed integer of equal length, if the value cannot be represented (6.2.1.2).
Bit string masked by the width of the conversion target type (with the upper bits truncated).

(17) The results of bitwise operations on signed integers (6.3).
Arithmetic shift is performed for a shift operator. For other operators, a signed integer is calculated as an unsigned value (as a bit image).

(18) The sign of the remainder on integer division (6.3.5).
The result of the "%" operator takes the sign of the first operand in the expression.

(19) The result of a right shift of a negative-valued signed integral type (6.3.7).
Arithmetic shift is performed.

(20) The representations and sets of values of the various types of floating-point numbers (6.1.2.5).
Refer to "4.1.3 Internal representation and value area of data".

(21) The direction of truncation when an integral number is converted to a floating-point number that cannot exactly represent the original value (6.2.1.3).
Rounded to the nearest representable direction.

(22) The direction of truncation or rounding when a floating-point number is converted to a narrower floating-point number (6.2.1.4).
Rounded to the nearest representable direction.

(23) The type of integer required to hold the maximum size of an array --- that is, the type of the sizeof operator, size_t (6.3.3.4, 7.1.1).
unsigned int type.

(24) The result of casting a pointer to an integer or vice versa (6.3.4).
Refer to "Specifying memory allocation area (__near /__far)" in "4.2.6 Using extended language specifications".

(25) The type of integer required to hold the difference between two pointers to members of the same array, ptrdiff_t (6.3.4, 7.1.1).
signed int type.

(26) The extent to which objects can actually be placed in registers by use of the register storage-class specifier (6.5.1).
User requests for register variables are not honored.

(27) A member of a union object is accessed using a member of a different type (6.3.2.3).
If the value of a union member is stored in a different member, the value will be stored in accordance with the alignment condition. As a result, when a union member is accessed using a member of a different type, the internal representation of the data will be of the type of the access.

(28) The padding and alignment of members of structures (6.5.2.1).
Refer to "4.1.3 Internal representation and value area of data".

(29) Whether a "plain" int bit-field is treated as a signed int bit-field or as an unsigned int bit-field (6.5.2.1).
Treated as an unsigned int type. However, this can be changed by option -signed_bitfield.

(30) The order of allocation of bit-fields within an int (6.5.2.1).
Allocated from the lower order.

(31) Whether a bit-field can straddle a storage-unit boundary (6.5.2.1).
When structure type packing is not specified, a bit-field cannot straddle a strage-unit boundary, but it is allocated to the next area.
When structure type packing is specified, a bit-field may straddle a strage-unit boundary.

(32) The integer type chosen to represent the values of an enumeration type (6.5.2.2).
Any of the char, signed char, unsigned char or signed short type. Minimum type that an enumerated type fits in.

(33) What constitutes an access to an object that has volatile-qualified type (6.5.3).
Although the access width, and order and number of accesses are as described in the C source, this does not apply to those accesses to a type for which the microcomputer does not have a corresponding instruction.

(34) The maximum number of declarators that may modify an arithmetic, structure, or union type (6.5.4).
128.

(35) The maximum number of case values in a switch statement (6.6.4.2).
65535.

(36) Whether the value of a single-character character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Whether such a character constant may have a negative value (6.8.1).
A value for the character constant specified in conditional inclusion is equal to the character constant value that appears in other expressions.
A character constant cannot be a negative value if it is a plain char type (char type which is neither signed nor unsigned) and a plain char type is unsigned. It can be a negative value if a plain char type is signed.

(37) The method for locating includable source files (6.8.2).
Folders are searched in this order and a file having the same name in the folder is identified as the header.
(1) Folder specified by the path (if it is full-path)
(2) Folder specified by option -I
(3) Standard include file folder (..\inc folder with a relative path from the bin folder where the compiler is placed)

(38) The support for quoted names for includable source files (6.8.2).
Searched in this order:
(1) Folder specified by the path (if it is full-path)
(2) Folder having the source file
(3) Folder specified by option -I
(4) Standard include file folder (..\inc folder with a relative path from the bin folder where the compiler is placed)

(39) The mapping of source file character sequences (6.8.2).
A character string described in the #include is interpreted as the character code specified as the source character set and is associated with a header name or an external source file name.

(40) The behavior on each recognized #pragma directive (6.8.6).
Refer to "4.2.4 #pragma directive".

(41) The definitions for __DATE__ and __TIME__ when respectively, the date and time of translation are not available (6.8.8).
A date and time are always obtained.

(42) The null pointer constant to which the macro NULL expands (7.1.6).
(void *)0.

(43) The diagnostic printed by and the termination behavior of the assert function (7.2).
The displayed diagnostic message is as follows:
Assertion failed : *expression*, file *file name*, line *line number*
The termination behavior depends on how the abort function is implemented.

(44) The sets of characters tested for by the isalnum, isalpha, iscntrl, islower, isprint, and isupper functions (7.3.1).
unsigned char type (0 to 255) and EOF (-1).

(45) The values returned by the mathematics functions on domain errors (7.5.1).
Refer to "7.5 Library Function".

(46) Whether the mathematics functions set the integer expression errno to the value of the macro ERANGE on underflow range errors (7.5.1).
ERANGE is set in errno in case of an underflow.

(47) Whether a domain error occurs or zero is returned when the fmod function has a second argument of zero (7.5.6.4).
A domain error is generated. For details, see the description about the fmod function group.

(48) The set of signals for the signal function (7.7.1.1).
The signal handling functions are not supported.

(49) The semantics for each signal recognized by the signal function (7.7.1.1).
The signal handling functions are not supported.

(50) The default handling and the handling at program startup for each signal recognized by the signal function (7.7.1.1).
The signal handling functions are not supported.

(51) If the equivalent of signal(sig, SIG_DFL); is not executed prior to the call of a signal handler, the blocking of the signal that is performed (7.7.1.1).
The signal handling functions are not supported.

(52) Whether the default handling is reset if the SIGILL signal is received by a handler specified to the signal function (7.7.1.1).
The signal handling functions are not supported.

(53) Whether the last line of a text stream requires a terminating new-line character (7.9.2).
The last line does not need to end in a newline character.

(54) Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.9.2).
Space characters appear when data is read.

(55) The number of null characters that may be appended to data written to a binary stream (7.9.2).
0.

(56) Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (7.9.3).
The file handling functions are not supported.

(57) Whether a write on a text stream causes the associated file to be truncated beyond that point (7.9.3).
The file handling functions are not supported.

(58) The characteristics of file buffering (7.9.3).
The file handling functions are not supported.

(59) Whether a zero-length file actually exists (7.9.3).
The file handling functions are not supported.

(60) The rules for composing valid file names (7.9.3).
The file handling functions are not supported.

(61) Whether the same file can be open multiple times (7.9.3).
The file handling functions are not supported.

(62) The effect of the remove function on an open file (7.9.4.1).
The file handling functions are not supported.

(63) The effect if a file with the new name exists prior to a call to the rename function (7.9.4.2).
The file handling functions are not supported.

(64) The output for %p conversion in the fprintf function (7.9.6.1).
Hexadecimal notation.
The fprintf function is not supported.

(65) The input for %p conversion in the fscanf function (7.9.6.2).
Hexadecimal number.
The fscanf function is not supported.

(66) The interpretation of a - character that is neither the first nor the last character in the scan list for %[conversion in the fscanf function (7.9.6.2).
Refer to "scanf" in "7.5.7 Standard I/O functions".
The fscanf function is not supported.

(67) The value to which the macro errno is set by the fgetpos or ftell function on failure (7.9.9.1, 7.9.9.4).
The file handling functions are not supported.

(68) The messages generated by the perror function (7.9.10.4).
Refer to "7.5 Library Function".

(69) The behavior of the calloc, malloc, or realloc function if the size requested is zero (7.10.3).
NULL is returned.

(70) The behavior of the abort function with regard to open and temporary files (7.10.4.1).
The file handling functions are not supported.

(71) The status returned by the exit function if the value of the argument is other than zero, EXIT_SUCCESS, or EXIT_FAILURE (7.10.4.3).
Not defined because of a freestanding environment.

(72) The set of environment names and the method for altering the environment list used by the getenv function (7.10.4.4).
The getenv function is not supported.

(73) The contents and mode of execution of the string by the system function (7.10.4.5).
The system function is not supported.

(74) The contents of the error message strings returned by the strerror function (7.11.6.2).
Refer to "7.5 Library Function".

(75) The local time zone and Daylight Saving Time (7.12.1).
time.h is not supported.

(76) The era for the clock function (7.12.2.1).
time.h is not supported.

Translation limits

The table below shows the translation limits of CC-RL.
The upper limit depends on the memory situation of the host environment for the item "No limit".

Table 4.1 Translation limits (C90)

Item	C90	CC-RL
Number of nesting levels of conditional inclusion	8	No limit
Number of pointers, arrays, and function declarators (in any combinations) qualifying an arithmetic, structure, union, or incomplete type in a declaration	12	128
Number of nesting levels of parenthesized declarators within a full declarator	31	No limit
Number of nesting levels of parenthesized expressions within a full expression	32	No limit
Number of significant initial characters in an internal identifier or a macro name	31	No limit
Number of significant initial characters in an external identifier	6	No limit
Number of external identifiers in one translation unit	511	No limit
Number of identifiers with block scope declared in one block	127	No limit
Number of macro identifiers simultaneously defined in one preprocessing translation unit	1024	No limit
Number of parameters in one function definition	31	No limit
Number of arguments in one function call	31	No limit
Number of parameters in one macro definition	31	No limit

Item	C90	CC-RL
Number of arguments in one macro invocation	31	No limit
Number of characters in a logical source line	509	No limit
Number of characters in a character string literal or wide string literal (after concatenation)	509	No limit
Number of bytes in an object (in a hosted environment only)	32767	32767(65535) Note 1
Number of nesting levels for #included files	8	No limit
Number of case labels for a switch statement (excluding those for any nested switch statements)	257	65535
Number of members in a single structure or union	127	No limit
Number of enumeration constants in a single enumeration	127	No limit
Number of levels of nested structure or union definitions in a single struct-declaration-list	15	No limit

Note 1. The value in parentheses indicates the number of bytes in cases where -large_variable is specified.

4.1.2 Implementation-defined behavior of C99

This section covers the implementation-defined behavior given by the C99 standard.

(1) How a diagnostic is identified (3.10, 5.1.1.3).
Refer to "10. MESSAGE".

(2) Whether each non-empty sequence of white-space characters other than new-line is retained or replaced by one space character in translation phase 3 (5.1.1.2).
Retained as they are.

(3) The mapping between physical source file multi-byte characters and the source character set in translation phase 1 (5.1.1.2).
Multibyte characters are mapped to the corresponding source character set according to the compile option.

(4) The name and type of the function called at program startup in a freestanding environment (5.1.2.1).
Not defined. Depends on the startup implementation.

(5) The effect of program termination in a freestanding environment (5.1.2.1).
Depends on startup in a normal termination. The abort function is used to terminate the program abnormally.

(6) An alternative manner in which the main function may be defined (5.1.2.2.1).
Not defined because of a freestanding environment.

(7) The values given to the strings pointed to by the argv argument to main (5.1.2.2.1).
Not defined because of a freestanding environment.

(8) What constitutes an interactive device (5.1.2.3).
Not defined for the configuration of an interactive device.

(9) The set of signals, their semantics, and their default handling (7.14).
The signal handling functions are not supported.

(10) Signal values other than SIGFPE, SIGILL, and SIGSEGV that correspond to a computational exception (7.14.1.1).
The signal handling functions are not supported.

(11) Signals for which the equivalent of signal(sig, SIG_IGN); is executed at program startup (7.14.1.1).
The signal handling functions are not supported.

(12) The set of environment names and the method for altering the environment list used by the getenv function (7.20.4.5).
The getenv function is not supported.

(13) The manner of execution of the string by the system function (7.20.4.6).
The system function is not supported.

(14) Which additional multibyte characters may appear in identifiers and their correspondence to universal character names (6.4.2).
Multibyte characters cannot be used as identifiers.

(15) The number of significant initial characters in an identifier (5.2.4.1, 6.4.2).
The entire identifier is handled as meaningful. The length of an identifier is unlimited.

(16) The number of bits in a byte (3.6).
8 bits.

(17) The values of the members of the execution character set (5.2.1).
The element values of the execution character set are ASCII code, EUC, SJIS, UTF-8, big5 and gb2312 values.

(18) The unique value of the member of the execution character set produced for each of the standard alphabetic escape sequences (5.2.2).

Escape Sequence	Value (ASCII)
"\a"	0x07
"\b"	0x08
"\f"	0x0C
"\n"	0x0A
"\r"	0x0D
"\t"	0x09
"\v"	0x0B

(19) The value of a char object into which has been stored any character other than a member of the basic execution character set (6.2.5).
Value that is type-converted to char type.

(20) Which of signed char or unsigned char has the same range, representation, and behavior as "plain" char (6.2.5, 6.3.1.1).
The char type has the same range of values, the same representation format and the same behavior as the unsigned char type. However, it can be switched to the signed char type by option -signed_char.

(21) The mapping of members of the source character set (in character constants and string literals) to members of the execution character set (6.4.4.4, 5.1.1.2).
Associated with the element having the same value.

(22) The value of an integer character constant containing more than one character or containing a character or escape sequence that does not map to a single-byte execution character (6.4.4.4).
A simple character constant consisting of up to two characters has a two-byte value with the lower byte being the last character and the upper byte being the start character. A character constant having three or more characters results in an error. A character which is not represented by basic execution environment character set is regarded as a simple character constant having that value. In an invalid backslash representation, the backslash is ignored and the next character is regarded as a simple character constant.

(23) The value of a wide character constant containing more than one multibyte character, or containing a multibyte character or escape sequence not represented in the extended execution character set (6.4.4.4).
Left-most character value as a multibyte character.

(24) The current locale used to convert a wide character constant consisting of a single multi-byte character that maps to a member of the extended execution character set into a corresponding wide character code (6.4.4.4).
Locale is not supported.

(25) The current locale used to convert a wide string literal into corresponding wide character codes (6.4.5).
Locale is not supported.

(26) The value of a string literal containing a multi-byte character or escape sequence not represented in the execution character set (6.4.5).
Corresponding byte value for escape sequence or corresponding each byte value for a multibyte character.

(27) Any extended integer types that exist in the implementation (6.2.5).
No extended integer types are provided.

(28) Whether signed integer types are represented using sign and magnitude, two's complement, or one's complement, and whether the extraordinary value is a trap representation or an ordinary value (6.2.6.2).
The signed integer type is represented in two's complement, and there are no trap representations.

(29) The rank of any extended integer type relative to another extended integer type with the same precision (6.3.1.1).
No extended integer types are provided.

(30) The result of, or the signal raised by, converting an integer to a signed integer type when the value cannot be represented in an object of that type (6.3.1.3).
Bit string masked by the width of the conversion target type (with the upper bits truncated).

(31) The results of some bit-wise operations on signed integers (6.5).
Arithmetic shift is performed for a shift operator. For other operators, a signed integer is calculated as an unsigned value (as a bit image).

(32) The accuracy of the floating-point operations and of the library functions in <math.h> and <complex.h> that return floating-point results (5.2.4.2.2).
Unknown.

(33) The rounding behaviors characterized by non-standard values of FLT_ROUNDS (5.2.4.2.2).
No nonstandard value is defined for FLT_ROUNDS.

(34) The evaluation methods characterized by non-standard negative values of FLT_EVAL_METHOD (5.2.4.2.2).
No nonstandard value is defined for FLT_EVAL_METHOD.

(35) The direction of rounding when an integer is converted to a floating-point number that cannot exactly represent the original value (6.3.1.4).
Rounded to the nearest representable direction.

(36) The direction of rounding when a floating-point number is converted to a narrower floating-point number (6.3.1.5).
Rounded to the nearest representable direction.

(37) How the nearest representable value or the larger or smaller representable value immediately adjacent to the nearest representable value is chosen for certain floating constants (6.4.4.2).
Rounded to the nearest value.

(38) Whether and how floating expressions are contracted when not disallowed by the FP_CONTRACT pragma (6.5).
Contraction of expressions depends on each option specification.
The FP_CONTRACT pragma does not work.
#pragma STDC FP_CONTRACT is ignored even if it is specified.

(39) The default state for the FENV_ACCESS pragma (7.6.1).
The default state of the FENV_ACCESS pragma is ON.
#pragma STDC FENV_ACCESS is ignored even if it is specified.

(40) Additional floating-point exceptions, rounding modes, environments, and classifications, and their macro names (7.6, 7.12).
As per the math.h library provided by the compiler. There are no additional definitions.

(41) The default state for the FP_CONTRACT pragma (7.12.2).
The default state of the FP_CONTRACT pragma is ON.

(42) Whether the "inexact" floating-point exception can be raised when the rounded result actually does equal the mathematical result in an IEC 60559 conformant implementation (F.9).
Floating-point exceptions are not supported.
No "inexact" floating-point exception is generated.

(43) Whether the underflow (and inexact) floating-point exception can be raised when a result is tiny but not inexact in an IEC 60559 conformant implementation (F.9).
Floating-point exceptions are not supported. No "underflow" or "inexact" floating-point exception is generated.

(44) The result of converting a pointer to an integer or vice versa (6.3.2.3).
Refer to "Specifying memory allocation area (__near /__far)" in "4.2.6 Using extended language specifications".

(45) The size of the result of subtracting two pointers to elements of the same array (6.5.6).
The resultant type is the signed int type.

(46) The extent to which suggestions made by using the register storage-class specifier are effective (6.7.1).
User requests for register variables are not honored.

(47) The extent to which suggestions made by using the inline function specifier are effective (6.7.4).
Inlining is always tried. However, inlining may not be performed depending on the condition.

(48) Whether a "plain" int bit-field is treated as signed int bit-field or as an unsigned int bit-field (6.7.2, 6.7.2.1).
Treated as an unsigned int type. However, this can be changed by option -signed_bitfield.

(49) Allowable bit-field types other than _Bool, signed int, and unsigned int (6.7.2.1).
All integer types are allowed.

(50) Whether a bit-field can straddle a storage-unit boundary (6.7.2.1).
When structure type packing is not specified, a bit-field cannot straddle a strage-unit boundary, but it is allocated to the next area.
When structure type packing is specified, a bit-field may straddle a strage-unit boundary.

(51) The order of allocation of bit-fields within a unit (6.7.2.1).
Allocated from the lower order.

(52) The alignment of non-bit-field members of structures (6.7.2.1).
Refer to "4.1.3 Internal representation and value area of data".

(53) The integer type compatible with each enumerated type (6.7.2.2).
Any of the char, signed char, unsigned char or signed short type. Minimum type that an enumerated type fits in.

(54) What constitutes an access to an object that has volatile-qualified type (6.7.3).
Although the access width, and order and number of accesses are as described in the C source, this does not apply to those accesses to a type for which the microcomputer does not have a corresponding instruction.

(55) How sequences in both forms of header names are mapped to headers or external source file names (6.4.7).
A character string described in the #include is interpreted as the character code specified as the source character set and is associated with a header name or an external source file name.

(56) Whether the value of a character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set (6.10.1).
A value for the character constant specified in conditional inclusion is equal to the character constant value that appears in other expressions.

(57) Whether the value of a single-character character constant in a constant expression that controls conditional inclusion may have a negative value (6.10.1).
A character constant cannot be a negative value if it is a plain char type (char type which is neither signed nor unsigned) and a plain char type is unsigned. It can be a negative value if a plain char type is signed.

(58) The places that are searched for an included < > delimited header, and how the places are specified other header is identified (6.10.2).
Folders are searched in this order and a file having the same name in the folder is identified as the header.
(1) Folder specified by the path (if it is full-path)
(2) Folder specified by option -I
(3) Standard include file folder (..\inc folder with a relative path from the bin folder where the compiler is placed)

(59) How the named source file is searched for in an included " " delimited header (6.10.2).
Searched in this order:
(1) Folder specified by the path (if it is full-path)
(2) Folder having the source file
(3) Folder specified by option -I
(4) Standard include file folder (..\inc folder with a relative path from the bin folder where the compiler is placed)

(60) The method by which preprocessing tokens (possibly resulting from macro expansion) in a #include directive are combined into a header name (6.10.2).
Treated as a preprocessing token of a single header or file name only in a macro that replaces preprocessing tokens with a single <character string> or "character string" format.

(61) The nesting limit for #include processing (6.10.2).
There are no limits.

(62) Whether the # operator inserts a \ character before the \ character that begins a universal character name in a character constant or string literal (6.10.3.2).
A \ character is not inserted in front of the first \ character.

(63) The behavior on each recognized non-STDC #pragma directive (6.10.6).
Refer to "4.2.4 #pragma directive" in the User's Manual.

(64) The definitions for __DATE__ and __TIME__ when respectively, the date and time of translation are not available (6.10.8).
A date and time are always obtained.

(65) Any library facilities available to a freestanding program, other than the minimal set required by clause 4 (5.1.2.1).
Refer to "7. LIBRARY FUNCTION SPECIFICATIONS".

(66) The format of the diagnostic printed by the assert macro (7.2.1.1).
As follows:
Assertion failed: *Expression*, function *function_name*, file *file_name*, line *line_number*

(67) The representation of the floating-point status flags stored by the fegetexceptflag function (7.6.2.2).
The fegetexceptflag function is not supported.

(68) Whether the feraiseexcept function raises the "inexact" floating-point exception in addition to the "overflow" or "underflow" floating-point exception (7.6.2.3).
The feraiseexcept function is not supported.

(69) Strings other than "C" and "" that may be passed as the second argument to the setlocale function (7.11.1.1).
The setlocale function is not supported.

(70) The types defined for float_t and double_t when the value of the FLT_EVAL_METHOD macro is less than zero or greater than two (7.12).
float_t is defined as the float type and double_t as the double type.

(71) Domain errors for the mathematics functions, other that those required by this International Standard (7.12.1).
A domain error occurs if:

- The input argument of the cos, sin, tan function groups is NaN or +/- Inf.
- The input argument of the atan, fabs, ceil, or floor function groups is NaN.
- Either input argument of the atan2 function group is NaN, or both arguments are +/- Inf.
- The input argument val of the frexp, modf function groups is NaN or +/- Inf.
- The input argument val of the ldexp function groups is NaN.
- The input argument x of the scalbn, scalbln function groups is NaN.
- Either argument of the fmod function group is NaN or +/- Inf.

For details, refer to (72).

(72) The values returned by the mathematics functions on domain errors (7.12.1).
The table below summarizes the conditions in which a domain error occurs and returned values.

functions	Domain error occurrence condition and return value
cos/cosf/cosl sin/sinf/sinl tan/tanf/tanl	Returns NaN if the input argument is NaN or +/- Inf.
atan/atanf/atanl fabs/fabsf/fabsl ceil/ceilf/ceill floor/floorf/floorl	Returns NaN if the input argument is NaN.
atan2/atan2f/atan2l	Returns NaN if either input argument is NaN or both input arguments are +/- Inf.
frexp/frexpf/frexpl	Returns NaN if the input argument val is NaN or +/- Inf, and returns 0 to argument *exp.
ldexp/ldexpf/ldexpl	Returns NaN if the input argument val is NaN.
scalbn/scalbnf/scalbnl scalbln/scalblnf/scalblnl	Returns NaN if the input argument x is NaN.
fmod/fmodf/fmodl	Returns NaN if the input argument x or y is NaN. Returns Nan if the input argument x is +/- Inf or the input argument y is 0. Returns x if the input argument x is not +/- Inf and the input argument y is +/- Inf.

functions	Domain error occurrence condition and return value
modf/modff/modfl	Returns 0 if the input argument val is +/- Inf, and returns +/- Inf to argument iptr. Returns NaN if the input argument val is NaN, and returns NaN to argument iptr.
Other functions	Returns NaN if a domain error occurs in accordance with the C99 standard.

(73) The values returned by the mathematics functions on underflow range errors, whether errno is set to the value of the macro ERANGE when the integer expression math_errhandling & MATH_ERRNO is nonzero, and whether the "underflow" floating-point exception is raised when the integer expression math_errhandling & MATH_ERREXCEPT is nonzero. (7.12.1).
The return value is 0 or a denormalized number. ERANGE is set in errno in case of an underflow. No "underflow" floating-point exception is generated.

(74) Whether a domain error occurs or zero is returned when an fmod function has a second argument of zero (7.12.10.1).
A domain error is generated. For details, see the description about the fmod function group.

(75) The base-2 logarithm of the modulus used by the remquo functions in reducing the quotient (7.12.10.3).
The remquo function group is not supported.

(76) Whether the equivalent of signal(sig, SIG_DFL); is executed prior to the call of a signal handler, and, if not, the blocking of signals that is performed (7.14.1.1).
The signal handling functions are not supported.

(77) The null pointer constant to which the macro NULL expands (7.17).
(void *)0.

(78) Whether the last line of a text stream requires a terminating new-line character (7.19.2).
The last line does not need to end in a newline character.

(79) Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.19.2).
Space characters appear when data is read.

(80) The number of null characters that may be appended to data written to a binary stream (7.19.2).
0.

(81) Whether the file position indicator of an append-mode stream is initially positioned at the beginning or end of the file (7.19.3).
The file handling functions are not supported.

(82) Whether a write on a text stream causes the associated file to be truncated beyond that point (7.19.3).
The file handling functions are not supported.

(83) The characteristics of file buffering (7.19.3).
The file handling functions are not supported.

(84) Whether a zero-length file actually exists (7.19.3).
The file handling functions are not supported.

(85) The rules for composing valid file names (7.19.3).
The file handling functions are not supported.

(86) Whether the same file can be simultaneously open multiple times (7.19.3).
The file handling functions are not supported.

(87) The nature and choice of encodings used for multibyte characters in files (7.19.3).
The file handling functions are not supported.

(88) The effect of the remove function on an open file (7.19.4.1).
The file handling functions are not supported.

(89) The effect if a file with the new name exists prior to a call to the rename function (7.19.4.2).
The file handling functions are not supported.

(90) Whether an open temporary file is removed upon abnormal program termination (7.19.4.3).
The file handling functions are not supported.

(91) Which changes of mode are permitted (if any), and under what circumstances (7.19.5.4).
The file handling functions are not supported.

(92) The style used to print an infinity or NaN, and the meaning of any n-char or n-wchar sequence printed for a NaN (7.19.6.1, 7.24.2.1).
inf or INF is output for a positive infinity, -inf or -INF for a negative infinity, and nan or NAN for a NaN.
n character strings or n wide character strings are not supported when a NaN is written.

(93) The output for %p conversion in the fprintf or fwprintf function (7.19.6.1, 7.24.2.1).
Hexadecimal notation.
The fprintf and fwprintf functions are not supported.

(94) The interpretation of a - character that is neither the first nor the last character, nor the second where a ^ character is the first, in the scanlist for %[conversion in the fscanf() or fwscanf() function (7.19.6.2, 7.24.2.1).
Refer to "scanf" in "7.5.7 Standard I/O functions".
The fscanf and fwscanf functions are not supported.

(95) The set of sequences matched by a %p conversion and the interpretation of the corresponding input item in the fscanf() or fwscanf() function (7.19.6.2, 7.24.2.2).
Hexadecimal number.
The fscanf and fwscanf functions are not supported.

(96) The value to which the macro errno is set by the fgetpos, fsetpos, or ftell functions on failure (7.19.9.1, 7.19.9.3, 7.19.9.4).
The file handling functions are not supported.

(97) The meaning of any n-char or n-wchar sequence in a string representing a NaN that is converted by the strtod(), strtof(), strtold(), wcstod(), wcstof(), or wcstold() function (7.20.1.3, 7.24.4.1.1).
Interpreted as a value other than a number of floating-point type in case of the strtod, strtof ot strtold function.
The wcstod, wcstof, and wcstold functions are not supported.

(98) Whether or not the strtod, strtof, strtold, wcstod, wcstof, or wcstold function sets errno to ERANGE when underflow occurs (7.20.1.3, 7.24.4.1.1).
The strtod, strtof and strtold functions set ERANGE in global variable errno.
The wcstod, wcstof, and wcstold functions are not supported.

(99) Whether the calloc, malloc, and realloc functions return a null pointer or a pointer to an allocated object when the size requested is zero (7.20.3).
NULL is returned.

(100) Whether open streams with unwritten buffered data are flushed, open streams are closed, or temporary files are removed when the abort or _Exit function is called (7.20.4.1, 7.20.4.4).
The file handling functions are not supported.

(101) The termination status returned to the host environment by the abort, exit, or _Exit function (7.20.4.1, 7.20.4.3, 7.20.4.4).
Not defined because of a freestanding environment.

(102) The value returned by the system function when its argument is not a null pointer (7.20.4.6).
The system function is not supported.

(103) The local time zone and Daylight Saving Time (7.23.1).
time.h is not supported.

(104) The range and precision of times representable in clock_t and time_t (7.23).
time.h is not supported.

(105) The era for the clock function (7.23.2.1).
time.h is not supported.

(106) The replacement string for the %Z specifier to the strftime, and wcsftime functions in the "C" locale (7.23.3.5, 7.24.5.1).
time.h is not supported.

(107) Whether or when the trigonometric, hyperbolic, base-e exponential, base-e logarithmic, error, and log gamma functions raise the "inexact" floating-point exception in an IEC 60559 conformant implementation (F.9).
No "inexact" floating-point exception is generated

(108) Whether the functions in <math.h> honor the rounding direction mode in an IEC 60559 conformant implementation (F.9).
The rounding direction mode is fixed.
The fesetround function is not supported.

(109) The values or expressions assigned to the macros specified in the headers <float.h>, <limits.h>, and <stdint.h> (5.2.4.2, 7.18.2, 7.18.3).
Refer to "Standard header" in "4.2.3 C99 language specifications supported in conjunction with C90".

(110) The number, order, and encoding of bytes in any object (when not explicitly specified in this International Standard) (6.2.6.1).
Refer to "4.1.3 Internal representation and value area of data".

(111) The value of the result of the sizeof operator (6.5.3.4).
Refer to "4.1.3 Internal representation and value area of data".

Translation limits

The table below shows the translation limits of CC-RL.
The upper limit depends on the memory situation of the host environment for the item "No limit".

Table 4.2 Translation limits (C99)

Item	C99	CC-RL
Number of nesting levels of blocks	127	No limit
Number of nesting levels of conditional inclusion	63	No limit
Number of pointers, arrays, and function declarators (in any combinations) qualifying an arithmetic, structure, union, or incomplete type in a declaration	12	128
Number of nesting levels of parenthesized declarators within a full declarator	63	No limit
Number of nesting levels of parenthesized expressions within a full expression	63	No limit
Number of significant initial characters in an internal identifier or a macro name	63	No limit
Number of significant initial characters in an external identifier	31	No limit
Number of external identifiers in one translation unit	4095	No limit
Number of identifiers with block scope declared in one block	511	No limit
Number of macro identifiers simultaneously defined in one preprocessing translation unit	4095	No limit
Number of parameters in one function definition	127	No limit
Number of arguments in one function call	127	No limit
Number of parameters in one macro definition	127	No limit
Number of arguments in one macro invocation	127	No limit
Number of characters in a logical source line	4095	No limit
Number of characters in a character string literal or wide string literal (after concatenation)	4095	No limit
Number of bytes in an object (in a hosted environment only)	65535	32767(65535)Note1
Number of nesting levels for #included files	15	No limit
Number of case labels for a switch statement (excluding those for any nested switch statements)	1023	65535
Number of members in a single structure or union	1023	No limit
Number of enumeration constants in a single enumeration	1023	No limit

Item	C99	CC-RL
Number of levels of nested structure or union definitions in a single struct-declaration-list	63	No limit

Note 1. The value in parentheses indicates the number of bytes in cases where -large_variable is specified.

4.1.3 Internal representation and value area of data

This section explains the internal representation and value area of each type for the data handled by the CC-RL.

(1) Integer type

(a) Internal representation

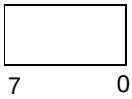
The leftmost bit in an area is a sign bit with a signed type. The value of a signed type is expressed as 2' s complement.

Figure 4.1 Internal Representation of Integer Type

_Bool

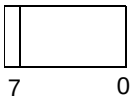
Only the 0th bit has meaning. Bits 1 to 7 are undefined.
When the -lang=c and -strict_std options are specified, _Bool type will cause a C90 violation error.

char

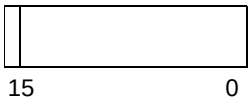


A plain char type not specified as signed or unsigned has the same representation as unsigned char.
When the -signed_char option is used, a plain char type has the same representation as signed char.

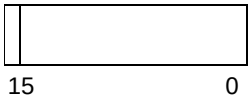
signed char (no sign bit for unsigned)



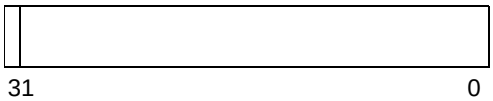
short (no sign bit for unsigned)



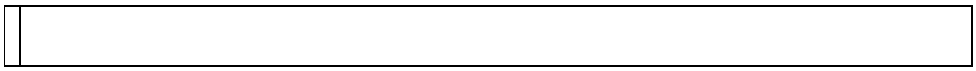
int (no sign bit for unsigned)



long (no sign bit for unsigned)



long long (no sign bit for unsigned)



When the -lang=c and -strict_std options are specified, long long type will cause a C90 violation error.

(b) Value area

Table 4.3 Value Area of Integer Type

Type	Value Area
_Bool	0 to 1
signed char	-128 to +127
signed short	-32768 to +32767
signed int	-32768 to +32767
signed long	-2147483648 to +2147483647
signed long long	-9223372036854775808 to +9223372036854775807
(unsigned) char	0 to 255
unsigned short	0 to 65535
unsigned int	0 to 65535
unsigned long	0 to 4294967295
unsigned long long	0 to 18446744073709551615

(c) Integer constants

The type of an integer constant will be the first type in the lists below capable of representing that value.

Table 4.4 Types of Integer Constants (If type long long is enabled (when -lang=c is specified and -strict_std is not))

Suffix	Decimal Constant	Binary Constant, Octal Constant, or Hexadecimal Constant
None	int long int unsigned long int[Note] long long int unsigned long long int	int unsigned int long int unsigned long int long long int unsigned long long int
u or U	unsigned int unsigned long int unsigned long long int	unsigned int unsigned long int unsigned long long int
l or L	long int unsigned long int[Note] long long int unsigned long long int	long int unsigned long int long long int unsigned long long int
Both u or U, and l or L	unsigned long int unsigned long long int	unsigned long int unsigned long long int
ll or LL	long long int unsigned long long int	long long int unsigned long long int
Both u or U, and ll or LL	unsigned long long int	unsigned long long int

Note Different from C99 specification. This is added to avoid the case where an integer constant represented as 4-byte data in C90 is unexpectedly represented as 8-byte data.

Table 4.5 Types of Integer Constants (If type long long is disabled (when -lang=c and -strict_std are specified))

Suffix	Decimal Constant	Binary Constant, Octal Constant, or Hexadecimal Constant
None	int long int unsigned long int	int unsigned int long int unsigned long int
u or U	unsigned int unsigned long int	unsigned int unsigned long int
l or L	long int unsigned long int	long int unsigned long int
Both u or U, and l or L	unsigned long int	unsigned long int

Table 4.6 Types of Integer Constants (If type long long is enabled (when -lang=c99 is specified))

Suffix	Decimal Constant	Binary Constant, Octal Constant, or Hexadecimal Constant
None	int long int long long int unsigned long long int	int unsigned int long int unsigned long int long long int unsigned long long int
u or U	unsigned int unsigned long int unsigned long long int	unsigned int unsigned long int unsigned long long int
l or L	long int long long int unsigned long long int	long int unsigned long int long long int unsigned long long int
Both u or U, and l or L	unsigned long int unsigned long long int	unsigned long int unsigned long long int
ll or LL	long long int unsigned long long int	long long int unsigned long long int
Both u or U, and ll or LL	unsigned long long int	unsigned long long int

(2) Floating-point type

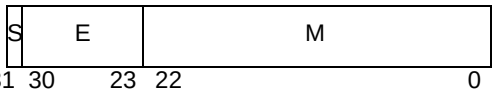
(a) Internal representation

Internal Representation of floating-point data type conforms to IEC 60559:1989 (IEEE 754-1985)[Note]. The leftmost bit in an area of a sign bit. If the value of this sign bit is 0, the data is a positive value; if it is 1, the data is a negative value.

Note IEEE: Institute of Electrical and Electronics Engineers
IEEE754 is a standard to unify specifications such as the data format and numeric range in systems that handle floating-point operations.

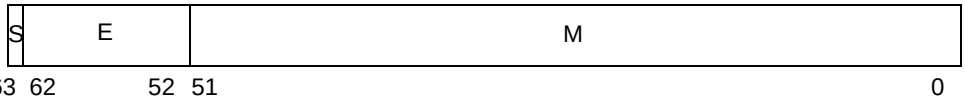
Figure 4.2 Internal Representation of Floating-Point Type

float



S: Sign bit of mantissa
E: Exponent (8 bits)
M: Mantissa (23 bits)

double, long double



S: Sign bit of mantissa
E: Exponent (11 bits)
M: Mantissa (52 bits)

When the option -dbl_size=4 is used, it has the same representation as type float. Even if you write type double, it will be treated as if you had written type float. Similarly, if you write long double, it will be treated as if you had written type float.
When the option dbl_size=8 is used, it is represented in 64 bits.

(b) Value area

Table 4.7 Value Area of Floating-Point Type

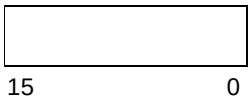
Type	Value Area
float	1.17549435E-38F to 3.40282347E+38F
double	2.2250738585072014E-308 to 1.7976931348623158E+308
long double	2.2250738585072014E-308 to 1.7976931348623158E+308

(3) Pointer type

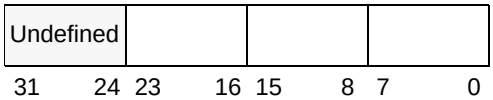
(a) Internal representation
The internal representation of a near pointer is a 16-bit unsigned type and that of a far pointer is a 32-bit unsigned type.
The most significant byte of a far pointer is undetermined.
The internal representation of a null pointer constant (NULL) has a value of 0. (Note that the byte corresponding to the undefined byte of the far pointer is not always 0.)
Therefore, for both the near and far pointers, access to address 0 is not guaranteed.
Correct operation is not guaranteed if the value of a far pointer exceeds 0xfffff.
Do not allocate a function or a variable to address 0x0f0000 or access the address.

Figure 4.3 Internal Representation of Pointer Type

near pointer



far pointer



(4) Enumerated type

(a) Internal representation
The internal representation of an enumerated type depends on the range of the enumerator value.

<1> When option -signed_char is not specified

Minimum Value of Enumerator	Maximum Value of Enumerator	Type of Internal Representation	Remark
-128	127	signed char	
0	255	char	When the range is 0 to 127, this type is used
Others		signed short	

<2> When option -signed_char is specified

Minimum Value of Enumerator	Maximum Value of Enumerator	Type of Internal Representation	Remark
-128	127	char	When the range is 0 to 127, this type is used
0	255	unsigned char	
Others		signed short	

(5) Array type

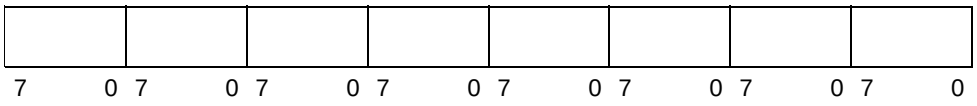
(a) Internal representation
The internal representation of an array type arranges the elements of an array in the form that satisfies the alignment condition (alignment) of the elements

Example

```
char    a[8] = {1, 2, 3, 4, 5, 6, 7, 8};
```

The internal representation of the array shown above is as follows.

Figure 4.4 Internal Representation of Array Type



(6) Structure type

(a) Internal representation
In a single structure, members are allocated from the head of the structure in the order of declaration. The internal representation of a structure type arranges the elements of a structure in a form that satisfies the alignment condition of the elements.
The alignment condition for the largest member of a structure is used as the alignment condition for the whole of the structure. This rule is also applied recursively when members are structures or unions.
The size of a structure is a multiple of the "alignment condition for the whole of the structure". Therefore, this size includes the unused area that is created to guarantee the alignment condition of the next data when the end of a structure does not match the alignment condition of that structure.

Example 1.

```
struct {
        short           s1;
        signed long     s2;
        char            s3;
        signed long     s4;
} s;
```

The internal representation of the structure shown above is as follows.

Figure 4.5 Internal Representation of Structure Type (without Structure Packing Specification)

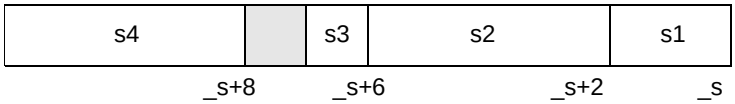
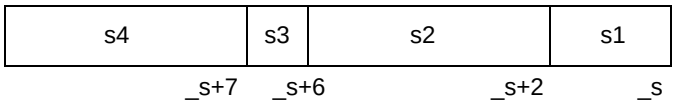


Figure 4.6 Internal Representation of Structure Type (with Structure Packing Specification)



Example 2.

```
struct {
        short           s1;
        char            s2;
} s;
```

The internal representation of the structure shown above is as follows.

Figure 4.7 Internal Representation of Structure Type (without Structure Packing Specification)

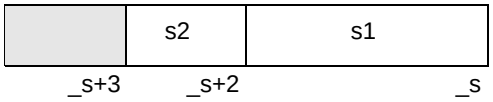
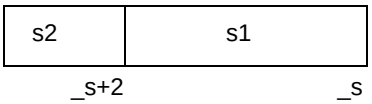


Figure 4.8 Internal Representation of Structure Type (with Structure Packing Specification)



For details on the structure packing specification, see "-pack".

(7) Union type

(a) Internal representation

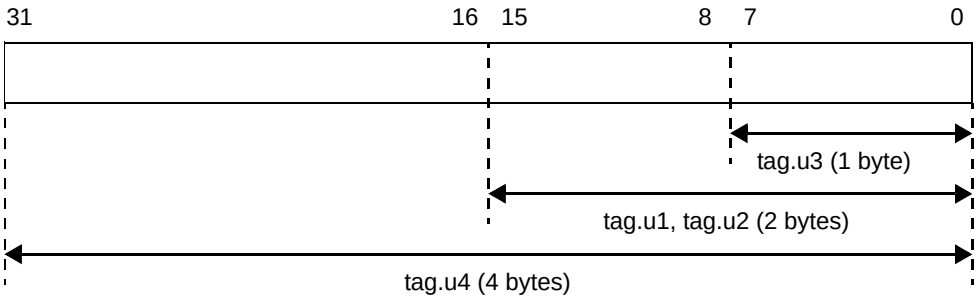
The alignment condition for the largest member of a union is used as the alignment condition for the whole of the union. This rule is also applied recursively when members are structures or unions.

Example

```
union {
        int     u1;
        short   u2;
        char    u3;
        long    u4;
} tag;
```

The internal representation of the union shown in the above example is as follows.

Figure 4.9 Internal Representation of Union Type



(8) Bit field

(a) Internal representation

The types that can be specified for bit fields are _Bool, char, signed char, unsigned char, signed short, unsigned short, signed int, unsigned int, signed long, unsigned long, signed long long, unsigned long long, and enumerated types.

Although only (signed or unsigned) int is allowed in C90, all of the above types are valid for bit fields in CC-RL if the -strict_std option is not used.

With the -strict_std option, the _Bool, signed long long, and unsigned long long types will cause a C90 violation error.

A bit field is allocated starting from the least significant bit for the type specified in the declaration of the bit field. When an attempt is made to allocate a bit field from the bit contiguous to the previous bit field and the location of the end of the bit field after allocation exceeds the location of adding the "bit width of declared type" to the boundary that is previous to satisfaction of the alignment condition for the bit field, the bit field is allocated to an area starting from the first boundary that satisfies the alignment condition after the previous bit field.

- The bit field of a type not specified as signed or unsigned is handled as unsigned.
  However, when the -signed_bitfield option is specified, it is handled as signed.
- Bit fields within an allocation unit are allocated from the lower order (LSB) toward the higher order (MSB).

Example 1.

```
struct S{
        char            a;
        char            b:2;
        signed char     c:3;
        unsigned char   d:4;
        int             e;
        short           f:5;
        int             g:6;
        unsigned char   h:2;
        unsigned int    i:2;
};
```

The internal representation for the bit field in the above example is as follows.

Figure 4.10 Internal Representation of Bit Field (without Structure Packing Specification)

sizeof(struct S)=8

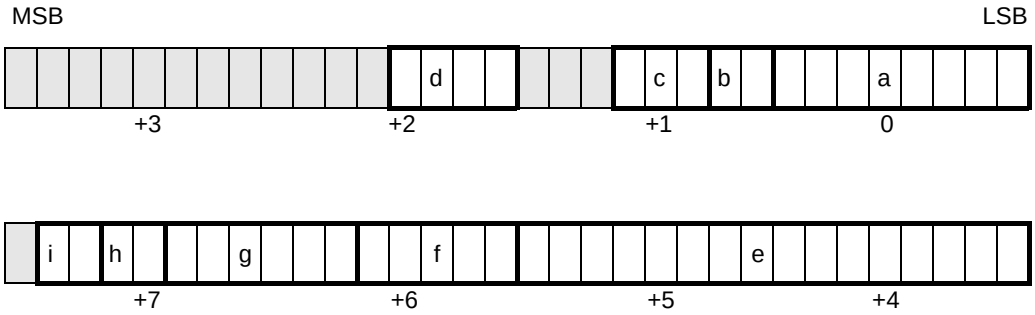
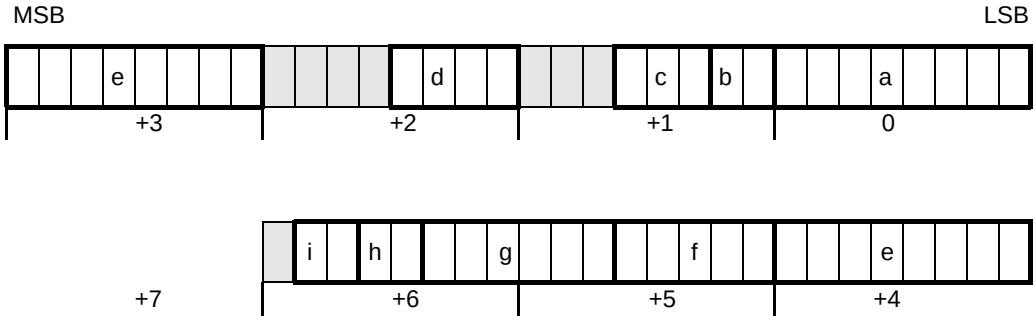


Figure 4.11 Internal Representation of Bit Field (with Structure Packing Specification)

sizeof(struct S)=7



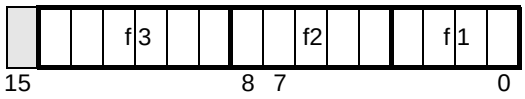
Example 2.

```
struct S{
        char    f1:4;
        int     f2:5;
        int     f3:6;
};
```

The internal representation for the bit field in the above example is as follows.

Figure 4.12 Internal Representation of Bit Field

sizeof(struct S)=2



Example 3.

```
struct S{
        long    f1:4;
};
```

The internal representation for the bit field in the above example is as follows.

Figure 4.13 Internal Representation of Bit Field (without Structure Packing Specification)

sizeof(struct S)=2

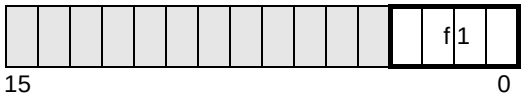
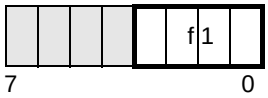


Figure 4.14 Internal Representation of Bit Field (with Structure Packing Specification)

sizeof(struct S)=1



For details on the structure packing specification, see "-pack".

(9) Alignment

(a) Alignment condition for basic type
Alignment condition for basic type is as follows.

Table 4.8 Alignment Condition for Basic Type

Basic Type	Alignment Conditions
(unsigned) char _Bool type	1-byte boundary
Others	2-byte boundary

(b) Alignment condition for enumerated type
Alignment condition for enumerated type is as follows.

<1> When option -signed_char is not specified

Minimum Value of Enumerator	Maximum Value of Enumerator	Type of Internal Representation	Alignment Condition
-128	127	signed char	1
0	255	char	1
Others		signed short	2

<2> When option -signed_char is specified

Minimum Value of Enumerator	Maximum Value of Enumerator	Type of Internal Representation	Alignment Condition
-128	127	char	1
0	255	unsigned char	1
Others		signed short	2

(c) Alignment condition for array type
The alignment condition for an array type is the same as that for the array elements.

(d) Alignment condition for pointers
The alignment condition (value) for near and far pointers is 2.

(e) Alignment condition for union type
The alignment conditions for a union type are the same as those of the structure's member whose type has the largest alignment condition.

(f) Alignment condition for structure type
The alignment conditions for a structure type are the same as those of the structure's member whose type has the largest alignment condition.

(g) Alignment condition for function argument
See "9.1 Function Call Interface".

(h) Alignment condition for function
The alignment condition for a function is a 1-byte boundary.

4.1.4 Allocation conditions for data and function

The allocation conditions for data and functions are as follows.

- 1Static variables and functions in a single section are allocated in the order of their definitions; that is, when data (or function) A and data (or function) B are allocated in the same section S and A is defined before B in a program, A is allocated to a smaller address than that of B in section S.
- Alignment of static variables and functions conforms to the alignment condition for the static variables and functions regardless of the type of the section; that is, even if a variable having an alignment value of 1 is allocated to a section having an alignment value of 2, the variable is aligned with a 1-byte boundary, and a variable having an alignment value of 2 cannot be allocated to a section having an alignment value of 1.

4.1.5 Static variable initialization

This section explains the static variable initialization.

4.1.5.1 Initialization by address calculation

When a static variable is initialized as follows, addition and subtraction affects only the lower-order 2 bytes.

((Integer type cast) Address constant) ± Integer type constant

This specification is valid only when -strict_std is not used.

Example

```
int     x;
static long     l = (long)&x + 1;       //The upper-order 2 bytes of &x do not change
```

4.1.5.2 Casting far address to near address and then converting to far address

When a static variable is initialized as follows, loss of the upper-order 2 bytes due to a cast to a near pointer does not occur.

(Cast to four or more bytes)(Cast to near pointer)(far address constant)

This specification is valid only when -strict_std is not used.

Example

```
int     __far x;
static long     l = (long)(int __near*)&x;      //far address is stored in l
```

4.2 Extended Language Specifications

This section explains the extended language specifications supported by the CC-RL.

4.2.1 Reserved words

The CC-RL adds the following characters as reserved words to implement the expanded function. These words are similar to the ANSI C keywords, and cannot be used as a label or variable name.
Reserved words that are added by the CC-RL are listed below.

Table 4.9 List of Reserved Words

Reserved Words	Usage
__saddr	Allocating a static variable to the saddr area
__callt	Calling a function with the callt instruction
__near	Specifying memory allocation area
__far	Specifying memory allocation area
__inline	Specifying memory allocation area
__sectop	Section start address
__secend	Section end address + 1

All names that include two underscores (__) are also invalid as label or variable names.

4.2.2 Macro

All the following macro names are supported.

Table 4.10 List of Supported Macros

Macro Name	Definition
__LINE__	Line number of source line at that point (decimal).
__FILE__	Name of source file (character string constant).
__DATE__	Date of translating source file (character string constant in the form of "Mmm dd yyyy"). Here, the name of the month is the same as that created by the asctime function stipulated by ANSI standards (3 alphabetic characters with only the first character is capital letter) (The first character of dd is blank if its value is less than 10).Note2
__TIME__	Translation time of source file (character string constant having format "hh:mm:ss" similar to the time created by the asctime function).Note2
__STDC__	Decimal constant 1 (defined when the -strict_std option is specified).Note1
__RENESAS__	Decimal constant 1.
__RENESAS_VERSION__	If the version is V.XX.YY.ZZ, this will be 0xXXYYZZ00. Example) V.1.00.00 -> -D__RENESAS_VERSION__=0x01000000
__RL78__	Decimal constant 1.
__RL78_S1__	Decimal constant 1 (defined when S1 is specified by the -cpu option).
__RL78_S2__	Decimal constant 1 (defined when S2 is specified by the -cpu option).
__RL78_S3__	Decimal constant 1 (defined when S3 is specified by the -cpu option).

Macro Name	Definition
__RL78_SMALL__	Decimal constant 1 (defined when small is specified by the -memory_model option or when S1 is specified by the -cpu option while the -memory_model option is not specified).
__RL78_MEDIUM__	Decimal constant 1 (defined when medium is specified by the -memory_model option or when S2 or S3 is specified by the -cpu option while the -memory_model option is not specified).
__CCRL__	Decimal constant 1.
__CCRL	Decimal constant 1.
__DBL4	Decimal constant 1 (defined when 4 is specified by the -dbl_size option).
__DBL8	Decimal constant 1 (defined when 8 is specified by the -dbl_size option).
__SCHAR	Decimal constant 1 (defined when the -signed_char option is specified).
__UCHAR	Decimal constant 1 (defined when the -signed_char option is not specified).
__SBIT	Decimal constant 1 (defined when the -signed_bitfield option is specified).
__UBIT	Decimal constant 1 (defined when the -signed_bitfield option is not specified).
__FAR_ROM__	Decimal constant 1 (defined when the -far_rom option is specified).
__CNV_CA78K0R__	Decimal constant 1 (defined when ca78k0r is specified by the -convert_cc option).
__CNV_NC30__	Decimal constant 1 (defined when nc30 is specified by the -convert_cc option).
__CNV_IAR__	Decimal constant 1 (defined when iar is specified by the -convert_cc option)
__BASE_FILE__	C source file name (character string constant). Unlike __FILE__, the C source file name is returned even when used in an include file.
__STDC_VERSION__	Decimal constant 199409L (defined when the -lang=c and -strict_std options are specified).Note1 Decimal constant 199901L (defined when the -lang=c99 option is specified).
__STDC_HOSTED__	Decimal constant 0 (defined when the -lang=c99 option is specified).
__STDC_IEC_559__	Decimal constant 1 (defined when the -lang=c99 option is specified).

Note 1. For the processing to be performed when the -strict_std option is specified, see "-strict_std [V1.06 or later] / -ansi [V1.05 or earlier]".

Note 2. The date and time of translation can be obtained in any case; the __DATE__ and __TIME__ macro values are always defined.

4.2.3 C99 language specifications supported in conjunction with C90

CC-RL supports some of the C99-standard specifications even when the C90 standard is selected (with -lang=c).

(1) Comment by //
Text from two slashes (//) until a newline character is a comment. If there is a backslash character (\) immediately before the newline, then the next line is treated as a continuation of the current comment.

(2) Concatenating wide string literal
The result of concatenating a character string literal with a wide string literal is a wide string literal.

(3) _Bool type
_Bool type is supported.
_Bool type is a 1-byte integer type that holds only 0 or 1.
When the -lang=c and -strict_std options are specified, _Bool type is not supported and it generates a compilation error.

(4) long long int type
long long int type is supported. long long int type is an 8-byte integer type.
Appending "LL" and "ULL" to a constant value is also supported. It is also possible to specify this for bit field types.
When the -lang=c and -strict_std options are specified, the long long int type is not supported and it generates a compilation error.

(5) Integer promotion
In accordance with support for types _Bool and long long, integer promotion is also in accordance with the C99 specification.
When the -lang=c and -strict_std options are specified, _Bool type and long long type is not supported and integer promotion is in accordance with the C90 specification.

(6) Aggregate initialization
The initializer for an aggregate or union type object that has automatic storage duration conforms to the C99 specifications.
In the C90 specifications, only a constant expression is allowed for an initializer, but other expressions can be used in CC-RL.

```
void func(int param) {
        int i = param;              //Allowed both in C90 and C99
        int array[] = { param };    //Not allowed in C90, allowed in C99, and
                                    //allowed in CC-RL
```

(7) Default argument promotions
In accordance with support for types _Bool and long long, default argument promotions is also in accordance with the C99 specification..

- Functions are called after expanding type _Bool_ to type int (2 bytes).
- Functions are called with type (unsigned) long long remaining as an 8 bytes value.
- When the option -dbl_size=4 is used, functions are called with type float remaining 4 bytes.
  This is because as a result of this option, even if a float is promoted to a double, the double type will have 4 bytes (same as the float type).

A near pointer is converted to a far pointer.
void* conforms to the rules for variable pointers.

(8) Comma permission behind the last enumerator of a enum definition
When defining an enum type, it is permissible for the last enumerator in the enumeration to be followed by a comma (,).

```
enum EE {a, b, c,};
```

When the -lang=c and -strict_std options are specified, this comma will generate an error.

(9) Types of integer constants
The type of an integer constant changes due to addition of the long long type. For details, see "(c) Integer constants" in "4.1.3 Internal representation and value area of data".

(10) Standard header
Standard header stdint.h is added, which defines the following types.

(a) limits.h

Table 4.11 limits.h

Name	Value	Meaning
CHAR_BIT	+8	The number of bits (= 1 byte) of the minimum object not in bit field
SCHAR_MIN	-128	Minimum value of signed char
SCHAR_MAX	+127	Maximum value of signed char
UCHAR_MAX	+255	Maximum value of unsigned char

Name	Value	Meaning
CHAR_MIN	0 (-128)	Minimum value of char (The default is the value of unsigned char. When the -signed_char option is specified, it becomes the value of signed char.)
CHAR_MAX	+255 (+127)	Maximum value of char (The default is the value of unsigned char. When the -signed_char option is specified, it becomes the value of signed char.)
SHRT_MIN	-32768	Minimum value of short int
SHRT_MAX	+32767	Maximum value of short int
USHRT_MAX	+65535	Maximum value of unsigned short int
INT_MIN	-32768	Minimum value of int
INT_MAX	+32767	Maximum value of int
UINT_MAX	+65535	Maximum value of unsigned int
LONG_MIN	-2147483648	Minimum value of long int
LONG_MAX	+2147483647	Maximum value of long int
ULONG_MAX	+4294967295	Maximum value of unsigned long int
LLONG_MIN	-9223372036854775807	Minimum value of long long int (Invalid when using option -lang=c and -strict_std)
LLONG_MAX	+9223372036854775807	Maximum value of long long int (Invalid when using option -lang=c and -strict_std)
ULLONG_MAX	+18446744073709551615	Maximum value of unsigned long long int (Invalid when using option -lang=c and -strict_std)

(b) float.h

The values in parentheses are for the case when the -dbl_size=4 option is used, which specifies sizeof(double) = sizeof(long double) = 4. -dbl_size=4 is the default setting in the CC-RL.

Table 4.12 float.h

Name	Value	Meaning
FLT_ROUNDS	+1	Rounding mode for floating-point addition. 1 for the RL78 family (rounding in the nearest direction).
FLT_EVAL_METHOD	0	Evaluation format of floating-point number (Invalid when using option -lang=c)
FLT_RADIX	+2	Radix of exponent (b)
FLT_MANT_DIG	+24	Number of numerals (p) with FLT_RADIX of floating- point mantissa as base
DBL_MANT_DIG	+53 (+24)	
LDBL_MANT_DIG	+53 (+24)	

Name	Value	Meaning
DECIMAL_DIG	+17 (+9)	Number of digits of a decimal number (q) that can round a floating-point number of p digits using radix b to a decimal number of q digits and then restore the floating-point number of p digits using radix b without any change (Invalid when using option -lang=c)
FLT_DIG	+6	Number of digits of a decimal number (q) that can round a decimal number of q digits to a floating-point number of p digits of the radix b and then restore the decimal number of q
DBL_DIG	+15 (+6)	
LDBL_DIG	+15 (+6)	
FLT_MIN_EXP	-125	Minimum negative integer (e_{min}) that is a normalized floating-point number when FLT_RADIX is raised to the power of the value of FLT_RADIX minus 1.
DBL_MIN_EXP	-1021 (-125)	
LDBL_MIN_EXP	-1021 (-125)	
FLT_MIN_10_EXP	-37	Minimum negative integer $\log_{10} b^{e_{min-1}}$ that falls in the range of a normalized floating-point number when 10 is raised to the power of its value.
DBL_MIN_10_EXP	-307 (-37)	
LDBL_MIN_10_EXP	-307 (-37)	
FLT_MAX_EXP	+128	Maximum integer (e_{max}) that is a finite floating-point number that can be expressed when FLT_RADIX is raised to the power of its value minus 1.
DBL_MAX_EXP	+1024 (+128)	
LDBL_MAX_EXP	+1024 (+128)	
FLT_MAX_10_EXP	+38	Maximum integer that falls in the range of a normalized floating-point number when 10 is raised to this power. $\log_{10} ((1 - b^{-p}) * b^{e_{max}})$
DBL_MAX_10_EXP	+308 (+38)	
LDBL_MAX_10_EXP	+308 (+38)	
FLT_MAX	3.40282347E + 38F	Maximum value of finite floating-point numbers that can be expressed $(1 - b^{-p}) * b^{e_{max}}$
DBL_MAX	1.7976931348623158E+308 (3.40282347E+38F)	
LDBL_MAX	1.7976931348623158E+308 (3.40282347E+38F)	
FLT_EPSILON	1.19209290E - 07F	Difference between 1.0 that can be expressed by specified floating-point number type and the lowest value which is greater than 1. b^{1-p}
DBL_EPSILON	2.2204460492503131E-016 (1.19209290E - 07)	
LDBL_EPSILON	2.2204460492503131E-016 (1.19209290E - 07F)	
FLT_MIN	1.17549435E - 38F	Minimum value of normalized positive floating-point number $b^{e_{min-1}}$
DBL_MIN	2.2250738585072014E-308 (1.17549435E - 38F)	
LDBL_MIN	2.2250738585072014E-308 (1.17549435E - 38F)	

(c) stdint.h

Table 4.13 Type Definition Names in stdint.h

Type Name	Actual Type	Remark
int8_t	signed char	
int16_t	signed short	
int32_t	signed long	
int64_t	signed long long	When -strict_std is not used or -lang=c99 is used
uint8_t	unsigned char	
uint16_t	unsigned short	
uint32_t	unsigned long	
uint64_t	unsigned long long	When -strict_std is not used or -lang=c99 is used
int_least8_t	signed char	
int_least16_t	signed short	
int_least32_t	signed long	
int_least64_t	signed long long	When -strict_std is not used or -lang=c99 is used
uint_least8_t	unsigned char	
uint_least16_t	unsigned short	
uint_least32_t	unsigned long	
uint_least64_t	unsigned long long	When -strict_std is not used or -lang=c99 is used
int_fast8_t	signed char	
int_fast16_t	signed short	
int_fast32_t	signed long	
int_fast64_t	signed long long	When -strict_std is not used or -lang=c99 is used
uint_fast8_t	unsigned char	
uint_fast16_t	unsigned short	
uint_fast32_t	unsigned long	
uint_fast64_t	unsigned long long	When -strict_std is not used or -lang=c99 is used
intptr_t	signed long	
uintptr_t	unsigned long	
intmax_t	signed long	When -lang=c and -strict_std are used
	signed long long	When -strict_std is not used or -lang=c99 is used

Type Name	Actual Type	Remark
uintmax_t	unsigned long	When -lang=c and -strict_std are used
	unsigned long long	When -strict_std is not used or -lang=c99 is used

Table 4.14 Macro Definition Names in stdint.h

Macro Name	Value	Remark
INT8_MIN	-128	
INT16_MIN	-32768	
INT32_MIN	-2147483648	
INT64_MIN	-9223372036854775808	When -strict_std is not used or -lang=c99 is used
INT8_MAX	+127	
INT16_MAX	+32767	
INT32_MAX	+2147483647	
INT64_MAX	+9223372036854775807	When -strict_std is not used or -lang=c99 is used
UINT8_MAX	+255	
UINT16_MAX	+65535	
UINT32_MAX	+4294967295	
UINT64_MAX	+18446744073709551615	When -strict_std is not used or -lang=c99 is used
INT_LEAST8_MIN	-128	
INT_LEAST16_MIN	-32768	
INT_LEAST32_MIN	-2147483648	
INT_LEAST64_MIN	-9223372036854775808	When -strict_std is not used or -lang=c99 is used
INT_LEAST8_MAX	+127	
INT_LEAST16_MAX	+32767	
INT_LEAST32_MAX	+2147483647	
INT_LEAST64_MAX	+9223372036854775807	When -strict_std is not used or -lang=c99 is used
UINT_LEAST8_MAX	+255	
UINT_LEAST16_MAX	+65535	
UINT_LEAST32_MAX	+4294967295	
UINT_LEAST64_MAX	+18446744073709551615	When -strict_std is not used or -lang=c99 is used
INT_FAST8_MIN	-128	

Macro Name	Value	Remark
INT_FAST16_MIN	-32768	
INT_FAST32_MIN	-2147483648	
INT_FAST64_MIN	-9223372036854775808	When -strict_std is not used or -lang=c99 is used
INT_FAST8_MAX	+127	
INT_FAST16_MAX	+32767	
INT_FAST32_MAX	+2147483647	
INT_FAST64_MAX	+9223372036854775807	When -strict_std is not used or -lang=c99 is used
UINT_FAST8_MAX	+255	
UINT_FAST16_MAX	+65535	
UINT_FAST32_MAX	+4294967295	
UINT_FAST64_MAX	+18446744073709551615	When -strict_std is not used or -lang=c99 is used
INTPTR_MIN	-2147483648	
INTPTR_MAX	+2147483647	
UINTPTR_MAX	+4294967295	
INTMAX_MIN	-2147483648	When -lang=c and -strict_std are used
	-9223372036854775808	When -strict_std is not used or -lang=c99 is used
INTMAX_MAX	+2147483647	When -lang=c and -strict_std are used
	+9223372036854775807	When -strict_std is not used or -lang=c99 is used
UINTMAX_MAX	+4294967295	When -lang=c and -strict_std are used
	+18446744073709551615	When -strict_std is not used or -lang=c99 is used
PTRDIFF_MIN	-32768	
PTRDIFF_MAX	+32767	
SIZE_MAX	+65535	

(11) Variadic macro
The variadic macro is enabled.

```
#define pf(form, ...) printf(form, __VA_ARGS__)

pf("%s %d\n", "string", 100);
                ↓
printf("%s %d\n", "string", 100);
```

4.2.4 #pragma directive

Below are #pragma directives supported as extended language specifications. These extended specifications can also be used with the _Pragma operator in C99. See "4.2.6 Using extended language specifications" for details.

Table 4.15 List of Supported #pragma Directive

#pragma directive	Definition
#pragma interrupt	Hardware interrupt handler
#pragma interrupt_brk	Software interrupt handler
#pragma section	Changing compiler output section name
#pragma rtos_interrupt	Interrupt handler for RTOS
#pragma rtos_task	Task function for RTOS
#pragma inline	Inline expansion of function
#pragma noinline	
#pragma inline_asm	Describing assembler instruction
#pragma address	Absolute address allocation specification
#pragma saddr	Using saddr area
#pragma callt	callt function
#pragma near	near function
#pragma far	far function
#pragma pack	Structure packing
#pragma unpack	
#pragma stack_protector	Generating a code for detection of stack smashing [Professional Edition only]
#pragma no_stack_protector	

4.2.5 Binary constants

Binary constants are supported in the CC-RL.

4.2.6 Using extended language specifications

This section explains using expanded specifications.

- Using saddr area (__saddr)
- callt function (__callt)
- Specifying memory allocation area (__near /__far)
- Specifying inline function (__inline)
- Section address operator (__sectop/__secend)
- Hardware interrupt handler (#pragma interrupt)
- Software interrupt handler (#pragma interrupt_brk)
- Changing compiler output section name (#pragma section)
- Interrupt handler for RTOS (#pragma rtos_interrupt)
- Task function for RTOS (#pragma rtos_task)
- Inline expansion of function (#pragma inline, #pragma noinline)

- Describing assembler instruction (#pragma inline_asm)
- Absolute address allocation specification (#pragma address)
- Using saddr area (#pragma saddr)
- callt function (#pragma callt)
- near/far function (#pragma near/#pragma far) [V1.05 or later]
- Structure packing (#pragma pack/#pragma unpack) [V1.05 or later]
- Generating a code for detection of stack smashing (#pragma stack_protector/#pragma no_stack_protector) [Professional Edition only] [V1.02 or later]
- Binary constants

Using saddr area (__saddr)

When declared with __saddr, external variables and static variables in a function are allocated to the saddr area.

[Function]

- Initialized variables are allocated to the .sdata section.
- Uninitialized variables are allocated to the .sbss section.
- Address reference always returns a near pointer.
- External variables and static variables in a function are allocated to the saddr area when they are declared with __saddr.
- These variables are handled in the same way as other variables in a C source file.

[Effect]

- Instructions that access the saddr area are shorter than those accessing the normal memory area and their object code also becomes smaller, leading to improved execution speed.

[Usage]

- Specify __saddr in the declaration of the variable to be allocated to the saddr area.
  In a function, only variables with the extern or static storage class specifier can be declared with __saddr.

```
__saddr variable-declaration;
extern __saddr  variable-declaration;
static __saddr  variable-declaration;
__saddr extern  variable-declaration;
__saddr static  variable-declaration;
```

[Restrictions]

- If __saddr is specified for a const type variable, a compilation error will occur.
- If __saddr is specified for a function, a compilation error will occur.
- If __saddr is specified for a variable which does not have static storage duration, a compilation error will occur.
- If __saddr and __near or __far are specified together, a compilation error will occur.

```
__saddr __near int      ni;     //Error
__saddr __far int       fi;     //Error
```

- If __saddr is used in a typedef declaration, a compilation error will occur.

```
typedef __saddr int     SI;     //Error
```

[Example]

The following shows a sample C source code.

```
__saddr int     hsmm0;          //.sbss
__saddr int     hsmm1=1;        //.sdata
__saddr int     *hsptr;         //hsptr is allocated to. sbss
                                //The pointed location is in a normal area.
void main(){
        hsmm0 -= hsmm1;
        hsptr = 0;
}
```

The following shows the declarations and section allocation in the assembly source code.

```
        .PUBLIC _hsmm0                  ;Declaration
        .PUBLIC _hsmm1                  ;Declaration
        .PUBLIC _hsptr                  ;Declaration

        .SECTION        .sbss,SBSS      ;Allocated to the .sbss section
        .ALIGN  2
_hsmm0:
        .DS     2
        .ALIGN  2
_hsptr:
        .DS     2

        .SECTION        .sdata,SDATA    ;Allocated to the .sdata section
        .ALIGN  2
_hsmm1:
        .DB2    0x0001
```

The following shows the code in the function.

```
        movw    ax, _hsmm0
        subw    ax, _hsmm1
        movw    _hsmm0, ax
        movw    _hsptr, #0x0000
```

[Remark]

- Difference between the __saddr keyword and #pragma saddr
  - The __saddr keyword cannot be used together with the __near or __far keyword, and a compilation error will occur if used so.
  - #pragma saddr can be specified for a variable with __near or __far keyword. #pragma saddr overrides the __near or __far specification without a warning.

callt function (__callt)

A function declared with __callt is called by the callt instruction.

[Function]

- A function declared with __callt (callt function) is called by the callt instruction.
  The callt instruction enables a function whose "start address of function definition" is stored in the area (0x80 to 0xBF) called as the callt instruction table to be called by a shorter code than a direct function call.
- The callt function is called by the callt instruction using a label name with "@_" added to the beginning of the function name.
  When the callt function is called at the end of a function, the callt instruction is not used to make the call in some cases.
- The called function is handled as a normal function in the C source program.
- The specification becomes __near, and address reference always returns a near pointer.
- The callt function is allocated to a section with the relocation attribute TEXT.
- The callt instruction table is allocated to the .callt0 section.

[Effect]

- The size of the object code becomes smaller because the function is called by a 2-byte call instruction.

[Usage]

- The __callt declaration is made in the module that defines the function.

```
__callt function-declaration;
extern __callt  function-declaration;
static __callt  function-declaration;
__callt extern  function-declaration;
__callt static  function-declaration;
```

[Restrictions]

- The callt function is allocated to addresses 0xC0 to 0xFFFF regardless of the memory model.
- When __callt is specified for other than a function declaration, a compilation error will occur.
- The number of callt functions is checked at linkage.
- __callt needs to be specified for all declarations of the target function.
  When there is a __callt specification for only some declarations, a compilation error will occur.

```
        void func(void);
__callt void func(void);          //Error
```

- __callt can be specified simultaneously with __near but a compilation error will occur when it is specified simultaneously with __far.

```
__callt void func1(void);         //Function address is set to near
__callt __near void func2(void);  //Function address is set to near
__callt __far void func3(void);   //Error
```

- When __callt is used in typedef, a compilation error will occur.

```
typedef __callt int    CI;        //Error
```

- When there is a #pragma specification for a function that is qualified with __callt in the function declaration, a compilation error will occur.

[Example]

The following shows a sample C source code.

```
__callt void func(void){
          :
}

void main(void){
        func();    //Call of callt function
          :
}
```

The following shows the section allocation and output codes in the assembly source code.

```
        .PUBLIC _func
        .PUBLIC _main
        .PUBLIC @_func
          :

        .SECTION        .text,TEXT
_func:
          :

        .SECTION        .textf,TEXTF
_main:
        callt   [@_func]
          :

        .SECTION        .callt0,CALLT0
@_func:
        .DB2    _func
          :
```

[Remark]

- Difference between the __callt keyword and #pragma callt
  - The __callt keyword cannot be used together with the __far keyword, and a compilation error will occur if used so.
  - #pragma callt handles even a function to which the __far keyword is added as if __callt was specified without a warning being output.

Specifying memory allocation area (__near /__far)

An allocating place of the function and a variable can be designated specifically by adding the __near or __far type qualifier when a function or variable declared.

Remark 1. A declaration without __near or __far is handled according to the default __near and __far determined by the memory model.

Remark 2. near, far, __near, and __far have the following meanings here.

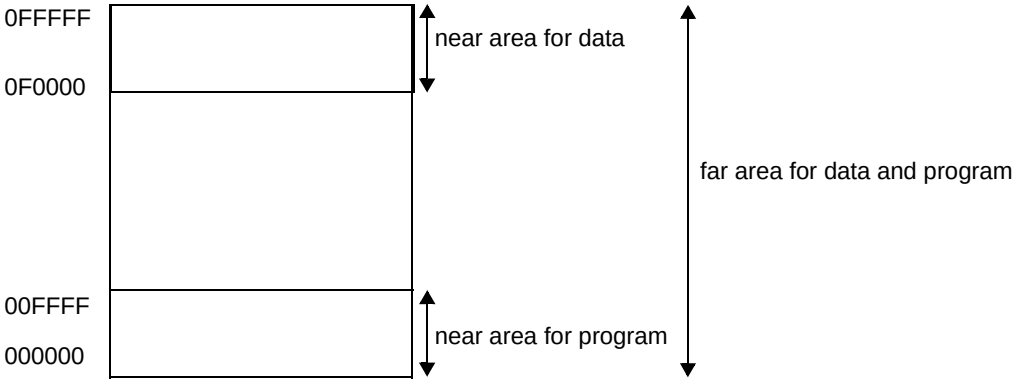
Character String	Meaning
near near area	Address range 0x0F0000 to 0x0FFFFF for RAM data and ROM data. Address range 0x000000 to 0x00FFFF for functions.
far far area	Address range 0x000000 to 0x0FFFFF for all RAM data, ROM data, and functions.
__near	Type qualifier indicating the near area.
__far	Type qualifier indicating the far area.

[Function]

- __near and __far are added as type qualifiers. Explicitly specifying __near or __far for a variable or function declaration gives the compiler a direction regarding the allocation area.
- __near and __far indicate that the variables and functions qualified by them are allocated to the following areas.

Type Qualifier	Function	Data
__near	0x000000 to 0x00FFFF	0x0F0000 to 0x0FFFFF
__far	0x000000 to 0x0FFFFF	0x000000 to 0x0FFFFF

Figure 4.15 Memory Image for near and far



- Explicit __near and __far specified in the source code take priority over the related option settings.
- For the internal representation of the pointer (2 bytes) pointing to the near area and that (4 bytes) pointing to the far area, see the "Pointer type" description.
- Extension from a pointer pointing to the near area to a pointer pointing to the far area is basically done as follows.

Pointer	Extension from near to far
Function pointer	The third byte from the least significant is set to 0x00. The most significant byte is undefined.
Variable pointer	The third byte from the least significant is set to 0x0f. The most significant byte is undefined.

For details of general cast specifications including integer types, see the "Cast" description.

- A pointer to void is handled as a variable pointer.
- The internal representation of a null pointer constant (NULL) is 0, both for a function pointer and a variable pointer. In a variable pointer pointing to far, the third byte from the least significant is set to 0x0f, but only in NULL, the pointer is set to 0x000000 (the most significant byte is undefined).
- Allocation sections for __near and __far

	Function	Uninitialized Variable	Initialized Non-const Variable	Initialized const Variable and Character String Data
__near	.text	.bss	.data	.const
__far	.textf	.bssf	.dataf	.constf

- Declaration
  - A declaration is checked from the right to the left, and when __near or __far is found between a "variable, function, or * " and the next " *, (, or left end of declaration", it is set to the __near or __far attribute for the "variable, function, or *".
    If neither __near nor __far is found, the default __near or __far attribute for the "variable, function, or * " is used.
    The order of __near/__far and other declarators can be changed as long as the above order is observed.

    The following shows examples in the medium model.

Declaration	Meaning
`int x;`	`int __near x;` `//int type variable x is //allocated to the near section`
`int func();`	`int __far func ();` `//Function func that returns a //int type is allocated to the //far section`
`int* x;`	`int __near * __near x;` `//Pointer x pointing to near is //allocated to the near section`
`int* func ();`	`int __near * __far func ();` `//Function func that returns //a near pointer is allocated //to the far section`
`int (*fp)();`	`int (__far * __near fp)();` `//far function pointer fp is //allocated to the near //section`
`int __far * func ();`	`int __far * __far func();` `//Function func that returns //a far pointer is allocated //to the far section`
`int (__far * fp)();`	`int (__far * __near fp)();` `//far function pointer fp is //allocated to the near //section`

- The following shows the locations of __near and __far specifications for variable declarations and their meaning.
  __near and __far are type qualifiers and can be written at the same locations as const and volatile. The objects to be affected by __near and __far are also the same as those of const and volatile.

```
int __far i;                    //i is output to the far section (.bssf)
                                //sizeof(&i) is 4
__near int j;                   //j is output to the near section (.bss)
                                //sizeof(&j) is 2
int __far * __near p;           //p is output to the near section (.bss)
                                //sizeof(&p) is 2
                                //p points to an int object in the far section
                                //sizeof(p) is 4
void (__far * __near fp)( );    //fp is output to the near section (.bss)
                                //sizeof(&fp) is 2
                                //fp points to a function in the far section
                                //sizeof(fp) is 4
```

- The following shows the locations of __near and __far specifications for function declarations and their meaning.
  __near and __far are type qualifiers and can be written at the same locations as const and volatile. The objects to be affected by __near and __far are the same as those of const and volatile.

```
void (__far func1)( );  //func1 is output to the far section (.textf)
                        //sizeof(&func1) is 4
void __far func2( );    //func2 is output to the far section (.textf)
                        //sizeof(&func2) is 4
```

- If multiple __near or __far qualifiers are specified in a single declaration, an error will occur.

```
int __near __far i;     //Error
int __near __near i;    //Error
int __far __far i;      //Error
```

- If both __near and __far are specified for declaration of a single variable or function, an error will occur.

```
__near int i;
__near int i;           //OK

__near int i;
__far int i;            //Compilation error

__near const int i;
       const int i;     //OK only when the default setting is near
                        //Compilation error when the default is far

       const int i;
__near const int i;     //OK only when the default setting is near
                        //Compilation error when the default is far
```

- If __near or __far is written as a structure or union member, an error will occur. [V1.04 or earlier]
- If __near or __far is written as a structure or union member, a warning is output and __near or __far will be ignored. [V1.05 or later]
- Relationship with keywords and #pragma
  For the operation when __near or __far is specified together with __saddr, __callt, #pragma callt, #pragma near, #pragma far, #pragma interrupt, or #pragma interrupt_brk, see the description of each keyword and "4.2.4 #pragma directive".

- Cast
  Cast involving a near pointer or a far pointer is handled as follows.
  - Variable pointer
    For conversion from near* to far*, 0x0f is set in the third byte from the least significant.

Note that only for NULL, both near* and far* are 0 and 0x00 is set in the third byte from the least significant.
For conversion between an integer type and a pointer type, the value is kept unchanged in principle.

		Converted to						
		_Bool	char	short int	near*	far*	long	long long
Converted from	_Bool char	-	-	-	<a>	<b>	-	-
	short int	-	-	-	<c>	<b>	-	-
	near*	<d>	<e>	<c>	-	<f>	<g>	<g>
	far*	<d>	<e>	<e>	<e>	-	<h>	<h>
	long	-	-	-	<e>	<c>	-	-
	long long	-	-	-	<e>	<e>	-	-

<a> After being extended to int, this type is converted to near* with the value unchanged.

<b> After being extended to long, this type is converted to far* with only the value of the lower-order three bytes unchanged.
(This is because the most significant byte is undefined and its value is not guaranteed.)

<c> This type is converted with the value unchanged. When it is converted to far*, the size of the type is set to three bytes and the upper-order bytes are truncated.
(This is because the most significant byte is undefined and its value is not guaranteed.)

<d> A NULL pointer is converted to 0. Other pointers are converted to 1.

<e> The upper-order bytes are truncated to fit within the size of the target type of conversion. The type is converted with the value of the remaining bytes unchanged. When the type is converted to far*, the size of the type is set to three bytes and the upper-order bytes are truncated.
(This is because the most significant byte is undefined and its value is not guaranteed.)

<f> For a NULL pointer, the third byte from the least significant is set to 0x00. For other pointers, the byte is set to 0x0F.
(This is because the most significant byte is undefined and its value is not guaranteed.)

<g> After being extended to a far pointer <f>, the upper-order bytes Note (including the undefined byte in the far pointer) are extended with zero.

<h> The upper-order bytes Note (including the undefined byte in the far pointer) are extended with zero.

Note This is applied to the most significant byte for conversion to the long type or the upper-order five bytes for conversion to the long long type.

- Function pointer
  For conversion from near* to far*, 0x00 is set in the third byte from the least significant.
  For conversion from far* to near*, the upper-order 2 bytes are truncated.
  Conversion is done in the same way as variable pointers except for conversion between pointers.
- Conversion between a variable pointer and a function pointer
  - When the -strict_std option is specified, type conversion between a variable pointer and a function pointer will cause an error. Note that explicit type conversion generates an error as well as implicit conversion.
  - When the -strict_std option is not used, conversion between a variable pointer and a function pointer is done with a warning being output.
    When the near or far specification is the same between the variable pointer and the function pointer, only the type is changed and no other processing is done because the size is the same before and after conversion.
    For conversion from far to near, the upper-order 2 bytes are discarded and truncated.
    For conversion from near to far, the pointer is extended from near to far with the type unchanged, and then the type is converted.

- Example of cast
  - Assignment between pointers

```
char __near* o_np;
char __far* o_fp;

typedef void(FT)(void);

__near FT* f_np;
__far FT* f_fp;

void func(){
    o_fp = o_np;  //Upper and lower-order 2 bytes of o_fp =(0xnn00, 0x0000)
                  //or (0xnn0f, o_np)
    f_fp = f_np;  //Upper and lower-order 2 bytes of f_fp =(0xnn00, f_np)
}
```

  - Conversion from an integer constant to "a variable pointer"

```
(char __near*)(char)0x12;           //0x0012
(char __near*)0x34;                 //0x0034
(char __far*)0x56;                  //0x00000056
(char __far*)(char __near*)0x78;    //0x000f0078
```

  - Conversion from an integer constant to "a function pointer"

```
typedef void(FT)(void);

void func1(){
  (__far FT*)0x34;                //0x00000034
  (__far FT*)(char __near*)0x56;  //0x000f0056
                                  //(char __near*)->(char __far*)->(__far FT*)
  (__far FT*)(char __far*)0x78;   //0x00000078
  (__far FT*)(__near FT*)0xab;    //0x000000ab
  (__far FT*)(__far FT*)0xcd;     //0x000000cd
}
```

  - Conversion from a variable to "a function pointer"

```
typedef void(FT)(void);

void func2(__near FT* f_np, unsigned int i, unsigned char ch){
        (__far FT*)f_np;        //0xnn00, f_np
        (__far FT*)i;           //0xnn00, i
        (__far FT*)ch;          //0xnn00, ch (Unsigned 2 bytes)
}
```

- Conversion from a variable to a pointer

```
signed char sc;
signed short ss;
signed long sl;

unsigned char uc;
unsigned short us;
unsigned long ul;

void func3(){
        (char __near*)uc;       //0x00, uc
        (char __near*)sc;       //(0x00 or 0xff), sc

        (char __far*)uc;        //nn, 0x00, 0x00, uc
        (char __far*)us;        //nn, 0x00, us(1), us(0)
        (char __far*)ul;        //nn, ul(2), ul(1), ul(0)

        (char __far*)sc;        //nn, (0x0000 or 0xffff), sc
        (char __far*)ss;        //nn, (0x00 or 0xff), ss(1), ss(0)
        (char __far*)sl;        //nn, sl(2), sl(1), sl(0)

        (__far FT*)uc;          //nn, 0x00, 0x00, uc
        (__far FT*)us;          //nn, 0x00, us(1), us(0)
        (__far FT*)ul;          //nn, ul(2), ul(1), ul(0)

        (__far FT*)sc;          //nn, (0x0000 or 0xffff), sc
        (__far FT*)ss;          //nn, (0x00 or 0xff), ss(1), ss(0)
        (__far FT*)sl;          //nn, sl(2), sl(1), sl(0)
}
```

- Conversion from a pointer to a variable

```
char __near* o_np;
char __far* o_fp;

typedef void(FT)(void);

__near FT* f_np;
__far FT* f_fp;

signed char sc;
signed short ss;
signed long sl;

unsigned char uc;
unsigned short us;
unsigned long ul;

void func(){
        uc = o_np;      //Least significant byte o_np
        uc = o_fp;      //Least significant byte o_fp
        uc = f_np;      //Least significant byte f_np
        uc = f_fp;      //Least significant byte f_fp

        us = o_np;      //Bit pattern of o_np is retained
        us = o_fp;      //Lower-order 2 bytes of o_fp
        us = f_np;      //Bit pattern of f_np is retained
        us = f_fp;      //Lower-order 2 bytes of f_fp

        ul = o_np;      //(0x0000,o_np)or(0x000f,o_np)
        ul = o_fp;      //(0x00,lower-order three bytes of o_fp,
                        //lower-order 2 bytes of o_fp)
        ul = f_np;      //(0x0000,f_np)
        ul = f_fp;      //(0x00,lower-order three bytes of of_fp,
                        //lower-order 2 bytes of f_fp)

        sc = o_np;      //Least significant byte o_np
        sc = o_fp;      //Least significant byte o_fp
        sc = f_np;      //Least significant byte f_np
        sc = f_fp;      //Least significant byte f_fp

        ss = o_np;      //Bit pattern of o_np is retained
        ss = o_fp;      //Lower-order 2 bytes of o_fp
        ss = f_np;      //Bit pattern of f_np is retained
        ss = f_fp;      //Lower-order 2 bytes of f_fp

        sl = o_np;      //(0x0000,o_np)or(0x000f,o_np)
        sl = o_fp;      //(0x00,lower-order three bytes of oo_fp,
                        //lower-order 2 bytes of o_fp)
        sl = f_np;      //(0x0000,f_np)
        sl = f_fp;      //(0x00,lower-order three bytes of of_fp,
                        //lower-order 2 bytes of f_fp)
}
```

- Pointer operation

  - Addition to a far pointer is done only in the lower-order 2 bytes. The upper-order bytes are not changed.

```
char __far* ptr = (char __far*)0x5ffff + 1;     //0x00050000
```

  - Subtraction to a far pointer is done only in the lower-order 2 bytes. The upper-order bytes are not changed.

```
char __far* ptr = (char __far*)0x050002 - 3;    //0x05ffff
```

- For subtraction between a near pointer and a far pointer, the type of the right term is cast to the type of the left term before subtraction.

```
__near int i;
__far int j;

void func( )
{
        &j - &i;        //OK (&j) - ((int __far *)(&i))
        &i - &j;        //OK (&i) - ((int __near *)(&j))

        &j - (__far int*)&i; //OK
}
```

- A near pointer is returned as the result of operation between a near pointer and an integer.

```
int b = ((char __near *)0xffff+1 == (char __near *)0);  //=1 (True)
```

- Comparison with pointers
  - When the -strict_std option is specified, a pointer cannot be directly compared with an integer. However, when the -strict_std option is not specified, direct comparison is done with a warning being output.
    In the case when a warning is output, the integer type is processed to match the pointer type before comparison. Conversion from an integer type to a pointer type conforms to the rules shown in the "Cast" description.
  - When the -strict_std option is specified, a variable pointer cannot be directly compared with a function pointer. However, when the -strict_std option is not specified, direct comparison is done with a warning being output.
    In the case when a warning is output, comparison is processed as follows.

  <a> The type of the right side is cast to the type of the left side.

  <b> When the near or far specification does not match between both sides, the near pointer is cast to the far pointer.

```
obj_near_ptr  == func_near_ptr  ->   obj_near_ptr == (obj *)func_near_ptr
func_near_ptr == obj_near_ptr   ->   func_near_ptr == (func *)obj_near_ptr
obj_near_ptr  == func_far_ptr
                      ->   (obj __far *)obj_near_ptr == (obj __far *)func_far_ptr
func_f_ptr == obj_n_ptr         ->   func_f_ptr == (func _far *)obj_n_ptr
func_n_ptr == obj_f_ptr
                       ->   (func __far *)func_n_ptr == (func __far *)obj_f_ptr
obj_f_ptr == func_n_ptr         ->   obj_f_ptr == (obj __far *)func_n_ptr
```

  - Equality operation (== or !=) for a far pointer is done only in the lower-order three bytes. The most significant byte does not affect the operation result.

```
if((char __far*)0x1105ffffUL == (char __far*)0x05ffffUL)
        OK();
else
        NG();
```

  - Relational operation (<,>,<=,>=) for a far pointer is done only in the lower-order 2 bytes. The upper-order 2 bytes do not affect the operation result.
    If comparison is to be performed including the upper-order bytes, they should be cast to unsigned long before comparison.
- Type of ptrdiff_t
  This type is always set to signed int (2-byte) regardless of the operation of near and far pointers.

[Usage]

- The __near or __far type qualifier is added to a function or variable declared.

[Example]

```
__near int i1;                          (1)
__far int i2;                           (2)
__far int *__near p1;                   (3)
__far int *__near *__far p2;            (4)
__far int func1( );                     (5)
__far int *__near func2 ( );            (6)
int (__near *__far fp1 ) ( );           (7)
__far int * (__near *__near fp2 ) ( );  (8)
__near int * (__far *__near fp3 ) ( );  (9)
__near int * (__near *__far fp4 ) ( );  (10)
```

(1) i1 has an int type and is allocated to the near area.

(2) i2 has an int type and is allocated to the far area.

(3) p1 is a 4-byte type variable that points to "an int type in the far area", and the variable itself is allocated to the near area.

(4) p2 is a 2-byte type variable that points to [a 4-byte type in the near area, which points to "an int type in the far area"], and the variable itself is allocated to the far area.

(5) func1 is a function that returns "an int type", and the function itself is allocated to the far area.

(6) func2 is a function that returns [a 4-byte type that points to "an int type in the far area", and the function itself is allocated to the near area.

(7) fp1 is a 2-byte type variable that points to [a function in the near area, which returns "an int type"], and the variable itself is allocated to the far area.

(8) fp2 is a 2-byte type variable that points to "a function in the near area, which returns [a 4-byte type that points to "an int type in the far area"]", and the variable itself is allocated to the near area.

(9) fp3 is a 4-byte type variable that points to "a function in the far area, which returns [a 2-byte type that points to "an int type in the near area"]", and the variable itself is allocated to the near area.

(10) fp4 is a 2-byte type variable that points to "a function in the near area, which returns [a 2-byte type that points to "an int type in the near area"]", and the variable itself is allocated to the far area.

Specifying inline function (__inline)

This notifies the compiler of an inline function.

[Function]

- The compiler inline-expands the function declared with __inline whenever possible.
- If both #pragma noinline and __inline are specified for the same function within a single translation unit, an error will occur.
- The conditions for the functions to be expanded inline depend on the -Oinline_level option setting.
  For the effect of this option, see the "-Oinline_level" description in the "2.5.1 Compile options".
- The function declared with __inline is expanded inline at the location from which the function is called.
- If the call and definition of the specified function differ in either of the following ways, a warning message will be output and inline specification will be ignored.
  - The numbers of parameters do not match.
  - The types of the return values or parameters do not match and type conversion is not possible.
    When type conversion is possible, the type is converted and then the function is expanded inline.
    In this case, the type of the return value is converted to that on the caller side and the types or parameters are converted to those in the function definition.
    Note that when the -strict_std option is specified, an error will occur even in this case.
- If the specified function satisfies any of the following conditions, inline specification will be ignored; no message will be output in this case.
  - The function has variable arguments.
  - The function calls itself during processing.
  - A call is done through the address of the function to be expanded inline.

[Effect]

- The function call and the code for returning to the caller are not output and execution is accelerated.
- As the inline-expanded code is optimized as well as the code before and after the function call, the effect of optimization may be improved.
- When the number of inline expansions is large, the code size may increase.

[Usage]

- Add __inline to the definition of a function.

[Example]

```
extern int      gi;
__inline int    i_func(int i)
{
        return ++i;
}

void func(int j)
{
        gi = i_func(j);
}
```

[Caution]

- Use of __inline does not guarantee inline expansion. The compiler may stop inline expansion depending on a limitation due to increased compilation time or memory usage
- When the -lang=c99 option is specified, the keyword __inline is an alias for the keyword inline; this is for compliance with the specification of inline for C99.

Section address operator (__sectop/__secend)

This is a section address operator.

[Function]

- The start address of *section-name* specified with __sectop is referenced.
- The end address of *section-name* +1 specified with __sectop is referenced.
- The return type of __sectop( ) and __secend( ) is void __far*.
- Addition and subtraction between the value obtained by __sectop or __secend and a constant should not be used.

[Usage]

```
__sectop("section-name")
__secend("section-name")
```

[Example]

The following shows a sample C source code.

```
const struct {
        void __far *rom_s;
        void __far *rom_e;
} DTBL[]={__sectop("XXX"), __secend("XXX")};
```

The following shows the declarations in the assembly source code.

```
        .PUBLIC         _DTBL
        .SECTION        .const, CONST
        .ALIGN          2
_DTBL:
        .DB2    LOWW(STARTOF(XXX))                      ;Lower-order 2 bytes of rom_s
        .DB     LOW(HIGHW(STARTOF(XXX)))                ;Least significant byte rom_s
        .DB     0x00
        .DB2    LOWW(STARTOF(XXX)+SIZEOF(XXX))          ;Lower-order 2 bytes of rom_e
        .DB     LOW(HIGHW(STARTOF(XXX)+SIZEOF(XXX)))    ;Least significant byte rom_e
        .DB     0x00
```

Hardware interrupt handler (#pragma interrupt)

Output object code required by hardware interrupt.

[Function]

- An interrupt vector is generated.
- A code for returning with RETI is generated with the target function definition used as a hardware interrupt handler.
- A stack or a register bank can be specified as the area for saving general registers.
- The interrupt handler definition is output to the .text or .textf section in the same way as normal function definitions. The section name can be changed through #pragma section.
  Note however when the vector table is specified, the start address of the interrupt handler should be a location that can be accessed in 16-bit addressing.
- When the vector table is specified, the specification becomes __near forcibly, regardless of whether __far is specified explicitly or implicitly. No warning message is output.
- When the vector table is not specified, the __near or __far specification by the function takes priority, regardless of whether the specification is explicit or implicit.

[Effect]

- Interrupt handlers can be written in the C source level.

[Usage]

- Specify a function name and interrupt specifications through the #pragma directive.
- Write the #pragma directive before the target function definition.

```
#pragma interrupt    [()]interrupt-handler-name[(interrupt-specification [,...])][()]
```

The interrupt specifications can include the following.

Item	Format	Description
Vector table specification	vect=*address*	Address of the vector table where the start address of the interrupt handler is to be stored. When this setting is omitted, no vector table is generated. Address of the vector table where the start address of the interrupt handler is to be stored. When this setting is omitted, no vector table is generated. *Address*: Binary, octal, decimal, or hexadecimal constant Only an even value within the range from 0x0 to 0x7c can be specified (a value outside this range will cause an error).
Register bank specification	bank={RB0|RB1|RB2|RB3}	Register bank for use by the interrupt handler.[Note 1] Execution of the interrupt handler uses the register in the specified bank. ES and CS are saved in the stack. Saving and restoring of general registers are implemented by using the SEL instruction that switches register banks. This can reduce the code size. When this setting is omitted, the general registers are saved in the stack.[Note 2]

Item	Format	Description
Nested interrupt enable specification	enable={true|false}	true: Enables nested interrupts at the entry to a function; that is, EI is generated before the codes of saving registers. false: EI is not generated. When enable is omitted, EI is not generated.

Note 1. Specify a register bank that differs from the register bank that was used before the interrupt handler was called. If the same register bank is specified, the contents of the saved register can no longer be restored.

Note 2. For details of register saving during interrupt processing or the stack frame, see "9.1 Function Call Interface".

If a single item is used more than once in a statement, a compilation error will occur.
However, more than one vector table can be specified as long as the addresses do not overlap. [V1.06 or later]

```
#pragma interrupt   func(vect=2,vect=2) // Error
```

```
#pragma interrupt   func(vect=2,vect=8) // OK
```

```
#pragma interrupt   func(vect=2)
#pragma interrupt   func(vect=8)        // OK
```

When "vect= address" is specified, a vector table is generated. Therefore, if a vector table is defined through Section definition directives in the assembly language (for example, in the startup routine), a linkage error will occur.
When "vect= address" is specified, use the .VECTOR directive to write a vector table in the assembly language instead of using the section definition directive.

[Restrictions]

- If an interrupt handler is called in the same way as a normal function, a compilation error will occur.
- If __inline, __callt or another #pragma is specified, a compilation error will occur.
- The parameters and return value of an interrupt handler should be declared as void (e.g., void func (void);).
  If the type is not void, a compilation error will occur.
- When no register is used or no function is called in an interrupt handler, the instruction for switching register banks is not output even if register bank switching is specified in #pragma interrupt.
  Examples of this situation include the following:
  - When only the instructions that do not use registers, such as instructions that specify constant values in SFR, are output in the interrupt handler

[Example]

(1) When there is no function call in the interrupt handler

(a) Default setting is used
[Input program]

```
#include "iodefine.h"                   /*Including iodefine.h enables*/
#pragma interrupt   inter (vect=INTP0)  /*interrupt source names to be used*/
                                        /*for vect setting*/

void inter ( void ) {
        /*Interrupt processing (only AX, HL, and ES are used)*/
}
```

[Output program]

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ;Assumed to be the __near specification
                                ;because the vector table is specified
_inter:
        push    AX              ;Code for saving the general registers to be
                                ;used in the stack
        push    HL
        mov     A, ES
        push    AX

        ;Interrupt processing for the INTP0 pin
        ;(body of the function. only AX, HL, and ES are used)

        pop     AX              ;Code for restoring the general registers to be
                                ;used from the stack
        mov     ES, A
        pop     HL
        pop     AX
        reti
```

(b) Register bank is specified

[Input program]

```
#pragma interrupt   inter (vect=INTP0, bank=RB1)

void inter ( void ) {
        /*Interrupt processing (ES is used but CS is not used)*/
}
```

[Output program]

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ;Assumed to be the __near specification
                                ;because the vector table is specified
_inter:
        sel     RB1             ;Code for switching register banks
        mov     A, ES           ;Code for saving ES and CS registers to be
                                ;used in the stack
        push    AX

        ;Interrupt processing for the INTP0 pin
        ;(body of the function. ES is used but CS is not used.)

        pop     AX              ;Code for restoring ES and CS registers to be
                                ;used from the stack
        mov     ES, A
        reti
```

(2) When a function is called in the interrupt handler (including a call of the function declared with #pragma inline_asm)

(a) Default setting is used

[Input program]

```
#pragma interrupt   inter (vect=INTP0)

void inter ( void ) {
        /*Interrupt processing*/
}
```

[Output program]

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ;Assumed to be the __near specification
                                ;because the vector table is specified
_inter:
        push    AX
        push    BC
        push    DE
        push    HL
        mov     A, ES           ;Code for saving ES, CS, and all general
                                ;registers in the stack
        mov     X, A
        mov     A, CS
        push    AX

        ;Interrupt processing for the INTP0 pin (body of the function)

        pop     AX              ;Code for restoring ES, CS, and all general
                                ;registers from the stack
        mov     CS, A
        mov     A, X
        mov     ES, A
        pop     HL
        pop     DE
        pop     BC
        pop     AX
        reti
```

(b) Register bank is specified

[Input program]

```
#pragma interrupt   inter (vect=INTP0, bank=RB1)

void inter ( void ) {
        /*Interrupt processing*/
}
```

[Output program]

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ;Assumed to be the __near specification
                                ;because the vector table is specified
_inter:
        sel     RB1             ;Code for switching register banks
        mov     A, ES           ;Code for saving ES and CS registers in the
                                ;stack
        mov     X, A
        mov     A, CS
        push    AX

        ;Interrupt processing for the INTP0 pin (body of the function)

        pop     AX              ;Code for restoring ES and CS registers in the
                                ;stack
        mov     CS, A
        mov     A, X
        mov     ES, A
        reti
```

Software interrupt handler (#pragma interrupt_brk)

Output object code required by software interrupt.

[Function]

- An interrupt vector is generated.
- A code for returning with RETB is generated with the target function definition used as a software interrupt handler.
- The interrupt handler definition is output to the .text section in the same way as normal function definitions. The section name can be changed through #pragma section.
  Note that the start address of the interrupt handler should be a location that can be accessed in 16-bit addressing.
- The specification becomes __near forcibly, regardless of whether __far is specified explicitly or implicitly.
  No warning message is output.

[Effect]

- Interrupt handlers can be written in the C source level.

[Usage]

- Specify a function name and interrupt specifications through the #pragma directive.
- Write the #pragma directive before the target function definition.

```
#pragma interrupt_brk   [()]interrupt-handler-name[(interrupt-specification
[,...])][()]
```

The interrupt specifications can include the following.

Item	Format	Description
Register bank specification	bank={RB0|RB1|RB2|RB3}	Register bank for use by the interrupt handler.Note 1 Execution of the interrupt handler uses the register in the specified bank. ES and CS are saved in the stack. Saving and restoring of general registers are implemented by using the SEL instruction that switches register banks. This can reduce the code size. When this setting is omitted, the general registers are saved in the stack.Note 2
Nested interrupt enable specification	enable={true|false}	true: Enables nested interrupts at the entry to a function; that is, EI is generated at the start of the interrupt handler generated by the compiler. false: EI is not generated. When enable is omitted, EI is not generated.

Note 1. Specify a register bank that differs from the register bank that was used before the interrupt handler was called. If the same register bank is specified, the contents of the saved register can no longer be restored.

Note 2. For details of register saving during interrupt processing or the stack frame, see "9.1 Function Call Interface".

If the same item is written more than once at the same time, a compilation error will occur.

Example

```
#pragma interrupt_brk func(bank=RB0, bank=RB1)     //Error
```

"#pragma interrupt_brk" generates a vector table. Therefore, if a vector table is defined through Section definition directives in the assembly language (for example, in the startup routine), a linkage error will occur.
Use the .VECTOR directive to write a vector table in the assembly language instead of using the section definition directive.

[Restrictions]

- If an interrupt handler is called in the same way as a normal function, a compilation error will occur.
- If __inline, __callt or another #pragma is specified, a compilation error will occur.
- The parameters and return value of an interrupt handler should be declared as void (e.g., void func (void);).
  If the type is not void, a compilation error will occur.
- If no register is used or no function is called in an interrupt handler, a register bank switching instruction is not output even though register bank switching has been specified by "#pragma interrupt".

[Example]

The output program is the same as that when #pragma interrupt is specified, except that the interrupt source is BRK and RETB is used in the code for returning.
Therefore, the following shows an example when there is no function call in the interrupt handler.

[Input program]

```
#pragma interrupt_brk   inter

void __near inter ( void ) {
        /*Interrupt processing (only AX, HL, and ES are used)*/
}
```

[Output program]

```
_inter  .vector 0x007E          ;BRK
        .SECTION    .text, TEXT
_inter:
        push    AX              ;Code for saving the general registers to be used in
                                ;the stack
        push    HL
        mov     A, ES
        push    AX

        ;Interrupt processing for the BRK instructions
        ;(body of the function. only AX, HL, and ES are used)

        pop     AX              ;Code for restoring the general registers to be used
                                ;from the stack
        mov     ES, A
        pop     HL
        pop     AX
        retb
```

Changing compiler output section name (#pragma section)

This changes a section name to be output by the compiler.

[Function]

- This changes a section name to be output by the compiler.
  For the section names output by the compiler and default allocation of the sections, see "6.1.1 Section name"
- This affects the declarations written after the #pragma directive.

[Effect]

- Assigning a new section name that differs from the default name to a specified set of data or functions allows this set of data or functions to be allocated to an address separate from that for other data or functions, and special processing can be applied in section units.

[Usage]

```
#pragma section section-type new-section-name
        section-type:{text|const|data|bss}
```

The section name of the specified type is changed.

```
#pragma section new-section-name
```

The section names of all types are changed.

```
#pragma section
```

The section names of all types are changed to their default section names.

- The characters shown below can be used in a new section name.
  The beginning of a name should be a character other than 0 to 9.
  "." can be used only to specify the section type and it can be used only at the beginning. If it is used in a character string not at the beginning, a compilation error will occur. If "." is used when the section type is not specified, a compilation error will occur.
  - 0 to 9
  - a to z
  - A to Z
  - .
  - @
  - _
- When the section type is text, the section name for the functions defined after the #pragma declaration is changed.
  Note that the start address of the interrupt handler should be a location that can be accessed in 16-bit addressing.
- When the section type is const, data, or bss, the name of the const, data, or bss section whose entity is defined after the #pragma declaration is changed.
- When both a section type and a new section name are specified, a character string is appended to the section name according to the following rules.
  - near section (.text, .const, .data, .bss)
    New section name + "_n"

```
#pragma section data MyData
int __near ndata = 5;           //Generated section name : MyData_n
```

- far section (.textf, .constf, .dataf, .bssf)
  New section name + "_f"

```
#pragma section bss MyBss
int __far fdata;                //Generated section name : MyBss_f
```

- saddr section (.sdata, .sbss)
  New section name + "_s"

```
#pragma section bss MyBss
int __saddr sdata;              //Generated section name : MyBss_s
```

	#pragma section text MyText	#pragma section bss MyBss	#pragma section data MyData	#pragma section const MyConst
void __near func() { }	MyText_n			
void __far func() { }	MyText_f			
int __near i;		MyBss_n		
int __far i;		MyBss_f		
int __saddr i;		MyBss_s		
int __near i = 5;			MyData_n	
int __far i = 5;			MyData_f	
int __saddr i = 5;			MyData_s	
const int __near i = 5;				MyConst_n
const int __far i = 5;				MyConst_f

- When only a new section name is specified, all section names for the program area, constant area, initialized data area, and uninitialized data area after the #pragma declaration are changed. In this case, each section name is created by appending the character string of the specified new section name at the end of the current section name and also post-fixing _s, _n, or _f. When the current section name is post-fixed with _n or _f, it is deleted.

```
#pragma section bss     XXX
int __near i;           //The section name is XXX_n
#pragma section YYY
int __far j;            //The section name is XXXYYY_f
                        //For bss, the section names change for both near and far.
                        //Therefore, XXX is added even for __far.
```

- When neither a section type nor a new section name are specified, all section names for the program area, constant area, initialized data area, and uninitialized data area after the #pragma declaration are restored to their default section names.
- When #pragma section is specified for a function, the scope of its effectiveness is not limited to within the function but extends to all locations following the specification. When a change to a text section is placed within a function, the scope is valid from the next function. A switch table is allocated to the section of the const data which have been specified at the start of the given function.

[Restrictions]

- The section name for an interrupt vector table cannot be changed.

[Example]

(1) To specify section names and section types

```
int __near ni_1;        //.bss
int __far fi_1;         //.bssf

#pragma section bss MyBss
int __near ni_2;        //MyBss_n
int __far fi_2;         //MyBss_f

#pragma section
int __near ni_3;        //.bss
int __far fi_3;         //.bssf
```

(2) To omit section types

```
#pragma section abc
int __near na;                  //Allocated to the .bssabc_n section
int __far fa;                   //Allocated to the .bssfabc_f section
int __near ni=1;                //Allocated to the .dataabc_n section
int __far fi=1;                 //Allocated to the .datafabc_f section
const int __near nc=1;          //Allocated to the .constabc_n section
const int __far fc=1;           //Allocated to the .constfabc_f section
void __near f(void)             //Allocated to the .textabc_n section
{
        na=nc;
}

#pragma section
int __near nb;                  //Allocated to the .bss section
void __near g(void)             //Allocated to the .text section
{
        nb=nc;
}
```

(3) To use both specifications with a section type and without a section type

```
int __near ni_1;                //.bss
int __far fi_1;                 //.bssf

#pragma section bss MyBss
int __near ni_2;                //MyBss_n
int __far fi_2;                 //MyBss_f

#pragma section XXX
int __near ni_3;                //Allocated to the MyBssXXX_n section
int __far fi_3;                 //Allocated to the MyBssXXX_f section

#pragma section
int __near ni_4;                //.bss
int __far fi_5;                 //.bssf
```

Interrupt handler for RTOS (#pragma rtos_interrupt)

The interrupt handler for Renesas RTOS for the RL78 family can be specified.

[Function]

- Interprets the function name specified with the #pragma rtos_interrupt directive as the interrupt handler for Renesas RTOS for the RL78 family.
- When the vector table is specified, the address of the described function name is registered in the specified interrupt vector table.
- The code for the body (function definition) of an interrupt handler is output to the .text or .textf section.
  The section name can be changed through #pragma section.
  Note however when the vector table is specified, the start address of the interrupt handler should be a location that can be accessed in 16-bit addressing.
- When the vector table is specified, the specification becomes __near forcibly, regardless of whether __far is specified explicitly or implicitly. No warning message is output.
- When the vector table is not specified, the __near or __far specification by the function takes priority, regardless of whether the specification is explicit or implicit.
- The interrupt handler for RTOS generates codes in the following order.

  (a) Calls kernel symbol __kernel_int_entry using call !!addr20 instruction
  When the vector table is specified, the interrupt address is passed to __kernel_int_entry as an argument.
  When the vector table is not specified, no argument is passed to __kernel_int_entry.

  (b) Allocates the local variable area (only when there is a local variable)

  (c) Executes the body of the function

  (d) Releases the local variable area (only when there is a local variable)

  (e) Unconditionally jumps to label _ret_int using br !!addr20 instruction

[Effect]

- The interrupt handler for RTOS can be described in the C source level.

[Usage]

- The interrupt address and function name is specified by the #pragma directive.

```
#pragma rtos_interrupt  [()function-name[(vect=address[Note])][)]
```

Note *address* : Binary, octal, decimal, or hexadecimal constant
Only an even value from 0 to 0x7c can be specified as the constant
(a value outside this range will cause an error).

- Write a #pragma directive before the function definition.

[Restrictions]

- If the function specified with #pragma rtos_interrupt is called in the same way as a normal function, a compilation error will occur.

```
#pragma rtos_interrupt func (vect=8)
void func(void) {}

void xxx()
{
        func();                 //Error
}
```

- The parameters and return value of a function should be declared as void (e.g., void func (void);).
  If the type is not void, a compilation error will occur.
- If an interrupt handler for RTOS is specified with __inline, __callt or another type of #pragma, a compilation error will occur.
- After the #pragma rtos_interrupt declaration line, a call or a definition of _kernel_int_exit or _kernel_int_entry as a function or a variable will generate an error.

```
#pragma rtos_interrupt func (vect=8)
void func(void) {
        _kernel_int_entry();    //Error
        _kernel_int_exit();     //Error
}

void    _kernel_int_entry(){}   //Error
void    _kernel_int_exit(){}    //Error
```

[Example]

The following shows an example in which the vector table is specified and no function is called in an interrupt handler.

[Input program]

```
#include "iodefine.h"

#pragma rtos_interrupt inthdr (vect=INTP0)
volatile int g;

void inthdr(void) {
        volatile int a;
        a = 1;
        g = 1;
}
```

[Output program]

```
        .SECTION .textf,TEXTF
_inthdr .vector 0x0008
_inthdr:
        push    ax                      ;ax register is saved
        movw    ax, #0x0008
        call    !!__kernel_int_entry    ;Registers other than ax are saved
        push    hl                      ;The area for local variables is allocated
        onew    ax
        movw    [sp+0x00], ax
        movw    !LOWW(_g), ax
        pop     hl                      ;The area for local variables is released
        br      !!__kernel_int_exit     ;Processing is specified so that all registers
                                        ;are restored when the interrupted task
                                        ;resumes execution
```

The following shows an example in which the vector table is not specified and no function is called in an interrupt handler.

[Input program]

```
#include "iodefine.h"

#pragma rtos_interrupt inthdr
volatile int g;

void inthdr(void) {
        volatile int a;
        a = 1;
        g = 1;
}
```

[Output program]

```
        .SECTION .textf,TEXTF
_inthdr:
        call    !!__kernel_int_entry    ;Registers other than ax are saved
        push    hl                      ;The area for local variables is allocated
        onew    ax
        movw    [sp+0x00], ax
        movw    !LOWW(_g), ax
        pop     hl                      ;The area for local variables is released
        br      !!__kernel_int_exit     ;Processing is specified so that all registers
                                        ;are restored when the interrupted task
                                        ;resumes execution
```

Task function for RTOS (#pragma rtos_task)

The function names specified with #pragma rtos_task are interpreted as the tasks for Renesas RTOS for the RL78 family.

[Function]

- The function names specified with #pragma rtos_task are interpreted as the tasks for RTOS.
- A task function can be coded without arguments specified, or with only one argument of signed long type specified, but no return values can be specified.
  An error will occur if two or more arguments are specified, an argument not of a signed long type is specified, or a return value is specified.
- No function exit processing is output for the task functions for RTOS.
- RTOS service call ext_tsk is called at the end of a task function for RTOS.
- RTOS service call ext_tsk calls the corresponding internal function of the OS through the br !!addr20 instruction. If ext_tsk is issued at the end of a normal function, no function exit processing is output.
- The code for the body (function definition) of a task function is output to the .text or .textf section. The section name can be changed through #pragma section.

[Effect]

- The task function for RTOS can be described in the C source level.
- As no function exit processing is output, the code efficiency is improved.

[Usage]

- Specifies the function name for the following #pragma directives.

```
#pragma rtos_task   [()]task-function-name [ ,...][)]
```

[Restrictions]

- If a task function for RTOS is specified with __inline or another type of #pragma, a compilation error will occur.
- If a task function for RTOS is called in the same way as a normal function, a compilation error will occur.

[Example]

The following shows a sample C source code.

```
#pragma rtos_task       func1
#pragma rtos_task       func2
extern void ext_tsk(void);
extern void g(int *a);

void func1 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        ext_tsk( );
}

void func2 ( signed long x ) {
        int a[3];
        a[0] = 1;
        g(a);
}

void func3 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        ext_tsk( );
}

void func4 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        if ( a[0] )
                ext_tsk( );
}
```

The following shows the assembly source output by compiler.

```
    .section        .textf, TEXTF
_func1 :
    subw    sp, #0x06             ;Stack is allocated
    onew    ax
    movw    [sp+0x00], ax         ;Assignment to an element of array a
    movw    ax, sp
    call    !!_g
    call    !!_ext_tsk            ;Function ext_tsk is called
    br      !!__kernel_task_exit  ;The epilogue for calling __kernel_task_exit is not
                                  ;output, which is always output by a task function
_func2 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax         ;Assignment to an element of array a
    movw    ax, sp
    call    !!_g
    br      !!__kernel_task_exit  ;The epilogue for calling __kernel_task_exit is not
                                  ;output, which is always output by a task function
_func3 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax         ;Assignment to an element of array a
    movw    ax, sp
    call    !!_g
    call    !!_ext_tsk            ;Function ext_tsk is called
    addw    sp, #0x06             ;The epilogue is output when #pragma rtos_task is
                                  ;not specified
    ret
_func4 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax         ;Assignment to an element of array a
    movw    ax, sp
    call    !!_g
    movw    ax, [sp+0x00]
    or      a, x
    bnz     $.BB@LABEL@4_2
.BB@LABEL@4_1:                    ;return
    addw    sp, #0x06             ;The epilogue is output when #pragma rtos_task is
                                  ;not specified
    ret
.BB@LABEL@4_2:                    ;bb3
    call    !!_ext_tsk
    br      $.BB@LABEL@4_1
```

Inline expansion of function (#pragma inline, #pragma noinline)

This notifies the compiler of an inline function.

[Function]

- #pragma inline declares a function to be expanded inline.
- #pragma noinline declares a function whose inline expansion is to be stopped when the -Oinline_level option is used.
- If both #pragma inline and #pragma noinline are specified for the same function within a single translation unit, an error will occur.
- The #pragma inline directive should be written before the definition of the target function within the translation unit that includes the function definition.
- #pragma inline has the same function as keyword __inline. For details of inline expansion, see "Specifying inline function (__inline)".

[Usage]

- #pragma inline and #pragma noinline are declared before the target functions.

```
#pragma inline      [(]function-name [,...][)]
#pragma noinline    [(]function-name [,...][)]
```

[Example]

```
extern int      gi;

#pragma inline  i_func

static int i_func(int i)
{
        return ++i;
}

void func(int j)
{
        gi = i_func(j);
}
```

Describing assembler instruction (#pragma inline_asm)

This specifies inline expansion of a function written in the assembly language.

[Function]

- Performs inline expansion on functions coded in assembly and declared with #pragma inline_asm.
- The calling conventions for an inline function with embedded assembly are the same as for ordinary function calls.

[Usage]

- #pragma inline_asm is declared before the target functions.

```
#pragma inline_asm  [(]function-name [,...][)]
```

[Restrictions]

- #pragma inline_asm should be specified before the definition of the function body.
- An external definition is also generated for a function specified with #pragma inline_asm.
- The compiler passes the character strings written in the functions specified with #pragma inline_asm to the assembler without change.
- The codes written in assembly language are processed by the preprocessor. Therefore, special care must be taken when the same names as the instructions or registers used in the assembly language are defined as the names of macros with #define. When including iodefine.h, the function written in the assembly language should be written before inclusion of iodefine.h.
- Assembler control instructions are not usable in assembly code for functions specified as inline_asm. In addition, only the directives listed below are usable. Specifying any other directive will lead to an error.
  - data definition/area reservation directives (.DB/.DB2/.DB4/.DB8/.DS)
  - macro directives (.MACRO/.IRP/.REPT/.LOCAL/.ENDM)
  - externally defined directive (.PUBLIC) [V1.04 or later]
- In the .PUBLIC directive in the function specified with inline_asm, only the labels defined in the function specified with inline_asm can be used. Using any other labels will lead to errors.

When a label is written in an assembly-language function, labels having the same name are generated for the number of times the function is expanded inline. In this case, take any of the following actions.

- Use a local label written in the assembly language. A local label has a single name in the assembly-language code, but the assembler automatically converts it into separate names.
  See ".LOCAL" for local labels.
- Ensure that an external label is expanded only in one location.
  - When calling an assembly-language function within the current source file, define the assembly-language function as static and call it only from a single location. Do not obtain the address of the assembly-language function.
  - When not calling an assembly-language function within the current source file, code the function as an external function.

[Example]

The following shows a sample C source code.

```
#pragma inline_asm func
void func(int x)
{
    movw !_a, ax
}

#pragma inline_asm func1
static void __near func1(void)  // When calling within the current file a function
{                               // that includes an external label definition and is
                                // specified with inline_asm, code it as a static
                                // __near function.
    .PUBLIC _label1
    incw ax
_label1:
    decw ax
}

#pragma inline_asm func2
void func2(void)                // When not calling within the current file a
{                               // function that includes an external label
                                // definition and is specified with inline_asm,
                                // code it as an external function.
    .PUBLIC _label2
    decw ax
_label2:
    incw ax
}

void main(void){
    func(3);
    func1();                    // Calls the function that includes an external
                                // label definition and is specified with inline_asm.
}
```

The following shows the assembly source output by compiler.

```
    .SECTION .textf,TEXTF
_func:
    .STACK _func = 4
    ._line_top inline_asm
    movw !_a, ax
    ._line_end inline_asm
    ret
_func2:
    .STACK _func2 = 4
    ._line_top inline_asm
    .PUBLIC _label2
    decw ax
_label2:
    incw ax
    ._line_end inline_asm
    ret
_main:
    .STACK _main = 4
    movw ax, #0x0003
    ._line_top inline_asm   ; Expanded code of func
    movw !_a, ax            ;
    ._line_end inline_asm   ;
    ._line_top inline_asm   ; Expanded code of func1
    .PUBLIC _label1         ;
    incw ax                 ;
_label1:                    ;
    decw ax                 ;
    ._line_end inline_asm   ;
    ret
```

Absolute address allocation specification (#pragma address)

Declare #pragma address in the module in which the variable to be allocated in an absolute address is to be defined. Variables can be allocated to the arbitrary address.

[Function]

- The specified variable is allocated to the specified address.

[Effect]

- Variables can be allocated to desired addresses.

[Usage]

- Declare #pragma address in the module in which the variable to be allocated in an absolute address is to be defined.

```
#pragma address [(]variable-name=absolute-address[Note][,...][)]
```

Note *Absolute-address* : Effective address (binary, octal, decimal, or hexadecimal constant in C language)

[Restrictions]

- #pragma address should be specified before declaration of the target variable. #pragma address after a variable declaration has no effect (no warning is output).
- If an object that is not a variable is specified, an error will occur.
- If #pragma address is specified for a const-qualified variable, an error will occur. [V1.04 or earlier]

```
#pragma address i=0xf2000
const int       i = 0;          //Error
```

- If #pragma address is specified for a variable declared with an initial value, an error will occur. [V1.04 or earlier]
- If #pragma address is specified for a variable that is not a const-qualified variable and is declared with an initial value, an error will occur. [V1.05 or later]

```
#pragma address i=0xf2000
int     i = 0;                  //Error
```

- If multiple #pragma address directives are specified for a single variable, an error will occur.

```
#pragma address i=0xf2000
#pragma address i=0xf2000       //Error
int     i;
```

- If a single address is specified for separate variables or the addresses allocated to separate variables overlap, an error will occur.
- When #pragma address is declared for a variable that is explicitly or implicitly specified as near and if the specified absolute address is not in the range from 0x0F0000 to 0x0FFFFF, a compilation error will occur. If the specified absolute address is in the SFR area, a linkage error will occur. [V1.04 or earlier]
- When #pragma address is declared for a variable that is not a const-qualified variable and is explicitly or implicitly specified as near and if the specified absolute address is not in the range from 0x0F0000 to 0x0FFFFF, a compilation error will occur. If the specified absolute address is in the SFR area, a linkage error will occur. [V1.05 or later]

```
#pragma address n_i1=0xF0000
char    __near n_i1;            //Can be compiled
#pragma address n_i2=0xEFFFF
char    __near n_i2;            //Error
#pragma address n_i3=0xEFFFF
char    n_i3;                   //Error
                                //This is because bss is set to near regardless of
                                //whether -memory_model=small or medium
#pragma address f_i=0xEFFFF
char    __far f_i;              //Can be compiled
```

- When the address specified for a const variable that is explicitly or implicitly specified as near is not in the mirror source area, a linkage error will occur. [V1.05 or later]

```
#pragma address i=0xf3000
const   int i = 0;              //Error
```

[Example]

The following shows a sample C source code.

```
#pragma address (io=0x0ffe00)

int     io;                     //io is allocated to address 0x0ffe00
func(void){
        io = 0;
}
```

Variables are declared and allocated to sections as follows in the assembly code.

```
        .PUBLIC         _io

        .SECTION        .bss, BSS
        .ORG            0xFFE00
_io:
        .DS             2
```

The following code is output in the function.

```
clrw    ax
movw    !LOWW(_io), ax
```

[Remark]

- Even when #pragma address is specified, the volatile attribute is not automatically added to variables.
  The -volatile option can add the volatile attribute to all static variables, including those with #pragma address.
  To add the volatile attribute separately to certain variables, declare each variable with volatile appended.
- When allocating a variable to a specified address, consider alignment of the variable. Do not allocate a variable over a 64-Kbyte boundary.

Using saddr area (#pragma saddr)

This notifies the compiler of a variable that is to be assigned to the saddr area.

[Function]

- Initialized variables are allocated to the .sdata section.
- Uninitialized variables are allocated to the .sbss section.
- Address reference always returns a near pointer.
- External variables and static variables in a function are allocated to the saddr area when they are specified with #pragma saddr.
- #pragma saddr handles even a variable to which the __far keyword is added as if __near was specified without a warning being output.

[Effect]

- Instructions that access the saddr area are shorter than those accessing the normal memory area and their object code also becomes smaller, leading to improved execution speed.

[Usage]

- Declare #pragma saddr before the first declaration of a variable.

```
#pragma saddr [()variable-name[,...][)]
```

[Restrictions]

- If there are multiple declarations for the same variable and #pragma saddr is written at the location where the second or subsequent declaration takes effect, correct operation is not guaranteed.
- If another #pragma is specified, a compilation error will occur.

[Example]

```
#pragma saddr saddr_var
extern int saddr_var;

void func(void)
{
        saddr_var = 0;
}
```

[Remark]

- Difference between the __saddr keyword and #pragma saddr
  - The __saddr keyword cannot be used together with the __near or __far keyword, and a compilation error will occur if used so.
  - #pragma saddr handles even a variable to which the __near or __far keyword is added as if __saddr was specified without a warning being output.

callt function (#pragma callt)

This notifies the compiler of a callt function.

[Function]

- A function specified with #pragma callt is called by the callt instruction.
  The callt instruction enables a function whose "start address of function definition" is stored in the area (0x80 to 0xBF) called as the callt instruction table to be called by a shorter code than a direct function call.
- The callt function is called by the callt instruction using a label name with "@_" added to the beginning of the function name.
  When the callt function is called at the end of a function, the callt instruction is not used to make the call in some cases.
- The called function is handled as a normal function in the C source program.
- The specification becomes __near, and address reference always returns a near pointer.
- The callt function is allocated to a section with the relocation attribute TEXT.
- The callt instruction table is allocated to the .callt0 section.
- #pragma callt handles even a variable to which the __far keyword is added as if __near was specified without a warning being output.
- When #pragma callt, #pragma near, or #pragma far was specified together for the same function, the #pragma directives become valid in the priority order of #pragma callt > #pragma near > #pragma far.

[Effect]

- The size of the object code becomes smaller because the function is called by a 2-byte call instruction.

[Usage]

- Declare #pragma callt before the first declaration of a function.

```
#pragma callt [()function-name[,...][)]
```

[Restrictions]

- If there are multiple declarations for the same variable and #pragma callt is written at the location where the second or subsequent declaration takes effect, correct operation is not guaranteed. #pragma callt should be written before the declaration.
- If a #pragma directive other than #pragma near or #pragma far is specified for the same function, a compilation error will occur.
- If __inline and __callt are specified in the declaration of a target function of this #pragma directive, a compilation error will occur.

[Example]

```
#pragma callt callt_func1
extern void callt_func1(void);

void func1(void)
{
        callt_func1();
          :
}

#pragma callt callt_func2
extern void __far callt_func2(void);  // Becomes __near without a warning
                                      // due to the effect of #pragma callt
void func2(void)
{
        callt_func2();
          :
}
```

[Remark]

- Difference between the __callt keyword and #pragma callt
  - The __callt keyword cannot be used together with the __far keyword, and a compilation error will occur if used so.
  - #pragma callt handles even a function to which the __far keyword is added as if __callt was specified without a warning being output.

near/far function (#pragma near/#pragma far) [V1.05 or later]

This notifies the compiler of a function that is specified with __near/__far.

[Function]

- A function specified with #pragma near is called as a function specified with __near.
  Address reference of a function specified with #pragma near always returns a near pointer and the function is allocated to a section with the relocation attribute TEXT.
  For details on __near, refer to "Specifying memory allocation area (__near /__far)".
- A function specified with #pragma far is called as a function specified with __far.
  Address reference of a function specified with #pragma far always returns a far pointer and the function is allocated to a section with the relocation attribute TEXT.
  For details on __far, refer to "Specifying memory allocation area (__near /__far)".
- A function specified with #pragma near ignores the keyword without a warning being output even for a function to which the __callt or __far keyword is added.
  To use the function as a callt function, specify #pragma callt instead of #pragma near for the function.
- A function specified with #pragma far ignores the keyword without a warning being output even for a function to which the __callt or __near keyword is added.
- When #pragma callt, #pragma near, or #pragma far was specified together for the same function, the #pragma directives become valid in the priority order of #pragma callt > #pragma near > #pragma far.

[Usage]

- Declare #pragma near/#pragma far before the first declaration of a variable.

```
#pragma near [() function-name [,...][)]
#pragma far [() function-name [,...][)]
```

[Restrictions]

- If there are multiple declarations for the same variable and #pragma near/#pragma far is written at the location where the second or subsequent declaration takes effect, correct operation is not guaranteed. Declare #pragma near/#pragma far before the first declaration.
- If a #pragma directive other than #pragma callt, #pragma near, or #pragma far is specified for the same function, a compilation error will occur.
- If __inline is specified in the declaration of a target function of this #pragma directive, a compilation error will occur.

[Example]

```
#pragma near func1,func3
#pragma far  func2,func3         // #pragma near takes priority for func3.

extern void func1(void);         // Becomes __near without a warning due to
                                 // the effect of #pragma near.
extern void __near func2(void);  // Becomes __far without a warning due to
                                 // the effect of #pragma far.
extern void __callt func3(void); // Becomes __near without a warning due to
                                 // the effect of #pragma near.
                                 // __callt becomes invalid without a warning.

void main(void)
{
    func1( );
    func2( );
    func3( );
      :
}
```

[Remark]

- Difference between the __near/__far keyword and #pragma near/#pragma far
  - #pragma near/#pragma far can be specified for multiple functions at the same time.
  - The __far keyword cannot be used together with the __callt keyword or __near keyword, and a compilation error will occur if used so.
  - #pragma near/#pragma far invalidates the __callt/__near/__far keyword without a warning.

Structure packing (#pragma pack/#pragma unpack) [V1.05 or later]

This specifies packing of a structure.

[Function]

- Packing is performed for a structure that is declared at or after the location where #pragma pack was specified. The number of alignment for a structure member is set to 1.
- Packing is not performed for a structure that is declared at or after the location where #pragma unpack was specified.
- If the -pack option is specified simultaneously with #pragma unpack, #pragma unpack takes priority.

[Usage]

- Declare #pragma pack/#pragma unpack before the declaration of the structure.

```
#pragma pack
#pragma unpack
```

[Example]

```
#pragma pack
struct s1 {
  char a;
  int b;    // The number of alignment is set to 1.
} st1;
#pragma unpack
struct s2 {
  char a;
  int b;    // The number of alignment is not set to 1.
} st2;
```

[Restrictions]

- Correct operation is not guaranteed if there is a mixture of C source files with different packing specifications for the same structure.
- Correct operation is not guaranteed if a structure, union, or address of those members whose alignment condition has been changed from two bytes to one byte by #pragma pack is passed as an argument of a standard library function.
- Correct operation is not guaranteed if the address of a structure or union member whose alignment condition has been changed from two bytes to one byte by #pragma pack is passed to a pointer whose type has two bytes as the alignment condition and indirect reference to the pointer is performed.

Generating a code for detection of stack smashing (#pragma stack_protector/#pragma no_stack_protector) [Professional Edition only] [V1.02 or later]

This generates a code for detection of stack smashing at the entry and end of a function.

[Function]

- This allocates a 2-byte area just before the local variable area (in the direction towards address 0xFFFFF) at the entry to a function, and the value specified by num is stored. After that, the 2-byte area in which num was stored is checked for smashing at the end of the function. If smashing has occurred, the __stack_chk_fail function is called.
- The __stack_chk_fail function needs to be provided by the user.
  It cannot be specified as a static function.

```
void __far __stack_chk_fail(void) {
/* Processing to be executed when the stack is smashed */
}
```

- A code for detection of stack smashing is not generated for a function for which #pragma no_stack_protector has been specified regardless of the -stack_protector option and -stack_protector_all option.

[Effect]

- Stack smashing can be detected by software.

[Usage]

- Declare #pragma stack_protector/#pragma no_stack_protector before the first declaration of a variable.

```
#pragma stack_protector [(]function-name[(num=number)][,function-name[(num=number)]][,...][)]
#pragma no_stack_protector [(]function-name[,...][)]
```

[Restrictions]

- Specify an integer from 0 to 65535 for the number to be set in num. If "= number" is omitted, the compiler automatically specifies the integer value.
- If this option is used simultaneously with the -stack_protector option or -stack_protector_all option, the specification by #pragma stack_protector/#pragma no_stack_protector becomes valid.
- A compile error will occur when #pragma stack_protector and #pragma no_stack_protector are specified simultaneously for the same function within a single translation unit.
- A compile error will occur when __inline, or other #pragma directives are specified.

Binary constants

Binary constants can be written in a C source program.

[Function]

- A binary constant can be written at a location where integer constants can be written.

[Effect]

- When writing a constant in a bit string, a binary constant can be directly written without being converted into an octal or hexadecimal constant and the readability is improved.

[Usage]

- A binary constant is written in the following manner.

```
0b    Binary constant
0B    Binary constant
```

- After 0b or 0B, write a sequence of numbers 0 and 1.
- One "_" can be written between numbers.
- The value of a binary constant is calculated with 2 as the radix.
- The type of a binary constant is the same as an octal or hexadecimal constant.

[Example]

The following shows a sample C source code.

```
int i1, i2, i3;

i1 = 0b00101100;
i2 = 0b0001_1000;
i3 = 0B0_1_0_1_0_1_0_1_0_1_0_1_0_1_0_1
```

The object code output by the compiler becomes the same as shown below.

```
int i1, i2, i3;

i1 = 0x2c;
i2 = 0x18;
i3 = 0x5555;
```

[Caution]

- If the code includes a binary constant and the -strict_std option is specified, an error will occur. [V1.06 or later]

4.2.7 Intrinsic functions

CC-RL provides the following "intrinsic functions". The intrinsic functions can only be called from function definitions. The instructions that can be described as functions are as follows.

Table 4.16 Intrinsic Function

Intrinsic Function	Function	Format
__DI	Outputs a DI instruction.	void __DI(void);
__EI	Outputs a EI instruction.	void __EI(void);
__halt	Outputs a HALT instruction.	void __halt(void);
__stop	Outputs a STOP instruction.	void __stop(void);
__brk	Outputs a BRK instruction.	void __brk(void);
__nop	Outputs a NOP instruction.	void __nop(void);
__rolb	Rotates x to the left y times assuming that x has a size of eight bits. Operation is undefined for the case where the rotation count is greater than the size of the value. If the rotation count may be greater than the size, mask the count to not exceed the size.	unsigned char __rolb(unsigned char x, unsigned char y);
__rorb	Rotates x to the right y times assuming that x has a size of eight bits. Operation is undefined for the case where the rotation count is greater than the size of the value. If the rotation count may be greater than the size, mask the count to not exceed the size.	unsigned char __rorb(unsigned char x, unsigned char y);
__rolw	Rotates x to the left y times assuming that x has a size of 16 bits. Operation is undefined for the case where the rotation count is greater than the size of the value. If the rotation count may be greater than the size, mask the count to not exceed the size.	unsigned int __rolw(unsigned int x, unsigned char y);
__rorw	Rotates x to the right y times assuming that x has a size of 16 bits. Operation is undefined for the case where the rotation count is greater than the size of the value. If the rotation count may be greater than the size, mask the count to not exceed the size.	unsigned int __rorw(unsigned int x, unsigned char y);
__mulu	Executes unsigned multiplication between (unsigned int)x and (unsigned int)y and returns a 16-bit result.	unsigned int __mulu(unsigned char x, unsigned char y);
__mului	Executes unsigned multiplication between (unsigned long)x and (unsigned long)y and returns a 32-bit result.	unsigned long __mului(unsigned int x, unsigned int y);

Intrinsic Function	Function	Format

__mulsi	Executes signed multiplication between (signed long)x and (signed long)y and returns a 32-bit result.	signed long __mulsi(
		  signed int x,
		  signed int y
		);

__mulul	Executes unsigned multiplication between (unsigned long long)x and (unsigned long long)y and returns a 64-bit result. When the -lang=c and -strict_std options are specified, this function is not defined as an intrinsic function.	unsigned long long __mulul(
		  unsigned long x,
		  unsigned long y
		);

__mulsl	Executes signed multiplication between (signed long long)x and (signed long long)y and returns a 64-bit result. When the -lang=c and -strict_std options are specified, this function is not defined as an intrinsic function.	signed long long __mulsl(
		  signed long x,
		  signed long y
		);

__divui	Executes unsigned division between x and y and returns a 16-bit result.
	When divisor y is 0, 0xFFFF is returned.
		unsigned int __divui(
		  unsigned int x,
		  unsigned char y
		);

__divul	Executes unsigned division between x and y and returns a 32-bit result.
	When divisor y is 0, 0xFFFFFFFF is returned.
		unsigned long __divul(
		  unsigned long x,
		  unsigned int y
		);

__remui	Executes unsigned remainder operation between x and y and returns a 8-bit result.
	When divisor y is 0, the lower-order 8 bits of dividend x are returned.
		unsigned char __remui(
		  unsigned int x,
		  unsigned char y
		);

__remul	Executes unsigned remainder operation between x and y and returns a 16-bit result.
	When divisor y is 0, the lower-order 16 bits of dividend x are returned.
		unsigned int __remul(
		  unsigned long x,
		  unsigned int y
		);

__macui	Executes unsigned multiply-accumulate operation (unsigned long) x * (unsigned long) y + z, and returns a 32-bit result.	unsigned long __macui(
		  unsigned int x,
		  unsigned int y,
		  unsigned long z
		);

__macsi	Executes unsigned multiply-accumulate operation (signed long) x * (signed long) y + z, and returns a 32-bit result.	signed long __macsi(
		  signed int x,
		  signed int y,
		  signed long z
		);

__get_psw	Returns the contents of PSW.	unsigned char __get_psw(void);
__set_psw	Sets x to PSW.	void __set_psw(unsigned char x);

__set1	The set1 instruction is used to set bit y of the address indicated by x to 1.
	Only a constant from 0 to 7 can be specified for bit y and a compile error will occur when any other value is specified.
		void __set1(
		  unsigned char __near *x,
		  unsigned char y
		);

Intrinsic Function	Function	Format
__clr1	The clr1 instruction is used to clear bit y of the address indicated by x to 0. Only a constant from 0 to 7 can be specified for bit y and a compile error will occur when any other value is specified.	void __clr1(unsigned char __near *x, unsigned char y);
__not1	The not1 instruction (xor instruction for the saddr area) is used to invert bit y of the address indicated by x. Only a constant from 0 to 7 can be specified for bit y and a compile error will occur when any other value is specified.	void __not1(unsigned char __near *x, unsigned char y);

5. ASSEMBLY LANGUAGE SPECIFICATIONS

This chapter explains the assembly language specifications supported by the CC-RL assembler.

5.1 Description of Source

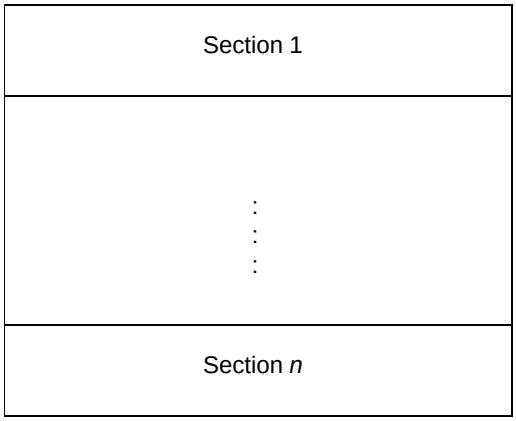
This section explains description of source, expressions, and operators.

5.1.1 Basic structure

This section explains the general structure of a source program.

(1) Section
A source program consists of blocks called sections.

Source program = A group of sections



There are two types of section; code sections and data sections.
Each section has a separate location counter. The location counter of a relocatable section holds an address relative to the start of the section, and that of an absolute section holds an absolute address in the memory space.
Each section can be allocated to any address through the optimizing linker, but the address of an absolute section cannot be changed from that specified in the source program.

(2) Module
A module is the unit of source program that is processed at one time by this assembler. One module corresponds to an assembly source file.

5.1.2 Description

An assembly source program consists of statements.
A statement is written in one line, using the characters listed in "(1) Character set". An assembly language statement consists of a "symbol", a "mnemonic", "operands", and a "comment".

```
[symbol][:]       [mnemonic]     [operand], [operand]    ;[comment]
```

These fields are delimited by a space, a tab, a colon (:), or a semicolon (;). The maximum number of characters in one line is theoretically 4294967294 (= 0xFFFFFFFE), but the memory size limits the actual maximum number of characters.
Statements can be written in a free format; as long as the order of the symbol, mnemonic, operands, and comment is correct, they can be written in any columns. Note that one statement can be written within one line.

To write a symbol in the symbol field, a colon, one or more spaces, or a tab should be appended to delimit the symbol from the rest of the statement. Whether colons or spaces or tabs are used, however, depends on the instruction coded by the mnemonic. Before and after a colon, any number of spaces or tabs can be inserted.

When operands are necessary, they should be separated from the rest of the statement by one or more spaces or tabs.

To write a comment in the comment field, it should be delimited from the rest of the statement by a semicolon. Before and after a semicolon, any number of spaces or tabs can be inserted.

One assembly language statement is described on one line. There is a line feed (return) at the end of the statement.

(1) Character set
The characters that can be used in a source program (assembly language) supported by the assembler are the following 3 types of characters.

- Language characters
- Character data
- Comment characters

(a) Language characters
These characters are used to code instructions in the source.
The language characters are further classified by their functions as follows.

Table 5.1 Language Characters and Character Set

General Name of Subclass		Character
Numerals		0 1 2 3 4 5 6 7 8 9
Alphabetic characters	Uppercase letter	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	Lowercase letter	a b c d e f g h i j k l m n o p q r s t u v w x y z
Characters similar to alphabet		@ _ (underscore) .(period)
Special characters	Special character type 1	. , : ; * / + - ` < > ( ) $ = ! & # [ ] " % << >> | ^ ? ~ Δ
	Special character type 2	\
	Special character type 3	LF, CR LF, HT

- The alphabetic characters (including the characters similar to alphabet) and numerals are collectively called alphanumeric characters.
- Reserved words and lowercase alphabetic characters specified in numeric constants are interpreted as the corresponding uppercase characters.
- When lowercase alphabetic characters are used in a user-defined symbol, the uppercase and lowercase are distinguished for interpretation.

The following shows the usage of special characters of type 1.
If a special character of this type appears outside constant data or comment fields in a source program for a purpose other than those listed below, an error will occur.

Table 5.2 Special Characters Type 1 and Usage of Characters

Character	Usage
.(period)	Bit position specifier Symbol for beginning a directive
, (comma)	Delimits an operand
: (colon)	Delimits a label Extended address specification ("ES:")
; (semicolon)	Beginning of comment
*	Multiplication operator
/	Division operator

Character	Usage
+	Positive sign Addition operator
- (hyphen)	Negative sign Subtraction operator
' (single quotation)	Symbol for beginning or ending a character constant
<	Relational operator Shift operator
>	Relational operator Shift operator
( )	Specifies an operation sequence
$	Symbol for beginning a control instruction Symbol specifying relative addressing Character similar to alphabet
=	Relational operator
!	Relational operator Beginning immediate addressing
&	Bit logic operator Logical operator
#	Beginning indicates Beginning comment (when used at the beginning of a line)
[ ]	Indirect indication symbol
"(double quotation)	Start and end of character string constant
%	Remainder operator
|	Bit logic operator Logical operator
^	Bit logic operator
?	Concatenation symbol (in macro body)
~	Bit logic operator
∆ (blank or tab)	Field delimiter

The following shows the usage of special characters of type 2.

Table 5.3 Special Characters Type 2 and Usage of Characters

Escape Sequence	Value (ASCII)	Meaning
\0	0x00	null character
\a	0x07	Alert (Warning tone)
\b	0x08	Backspace
\f	0x0C	Form feed (New Page)
\n	0x0A	New line (Line feed)
\r	0x0D	Carriage return (Restore)

Escape Sequence	Value (ASCII)	Meaning
\t	0x09	Horizontal tab
\v	0x0B	Vertical tab
\\	0x5C	Backslash
\'	0x27	Single quotation
\"	0x22	Double quotation
\?	0x3F	Question mark
ooo	0 - 0377	Octal number (0 to 255 in decimal) having up to three digits (o indicates an octal digit)
\x*hh*	0x00 - 0xFF	Hexadecimal number (0 to 255 in decimal) having up to two digits (*h* indicates a hexadecimal digit)

The following shows the usage of special characters of type 3.

<1> CR LF, LF
These characters delimit lines.

Special Character Type 3	Value Output to List
CR LF	0x0D0A
LF	0x0A

<2> HT
This character moves the column position in a source program. It is output to a list as is.

(b) Character data
Character data refers to characters used to write character constant, character string constant, and the quote-enclosed operands of some control instructions.

Caution Character data can use all characters (including multibyte character, although the encoding depends on the OS).

- Uppercase and lowercase characters are distinguished.
- The following shows the handling of HT, CR LF, and LF.

	Object Output	Value Output to Lis
HT	0x09	0x09 (a tab is expanded as is)
CR LF	0x0D0A	0x0D0A[Note]
LF	0x0A	0x0A[Note]

Note These characters only delimit lines and they are not regarded as part of the character data.

(c) Comment characters
Comment characters are used to write comments.

Caution Comment characters and character data have the same character set.

(2) Constants

A constant is a fixed value or data item and is also referred to as immediate data.
There are three types of constant as shown below.

- Numeric constants
- Character constants
- Character string constants

(a) Numeric constants

Integer constants can be written in binary, octal, decimal, or hexadecimal notation.
Integer constants has a width of 32 bits. A negative value is expressed as a 2's complement. If an integer value that exceeds the range of the values that can be expressed by 32 bits is specified, the assembler uses the value of the lower 32 bits of that integer value and continues processing (it does not output message).

Type	Notation	Example
Binary	Append an "0b" or "0B" suffix to the number. Append "b" or "B" at the end of the number.	0b1101, 0B1101 1101b, 1101B
Octal	Append an "0" suffix to the number. Append "o" or "O" at the end of the number.	074 074o, 074O
Decimal	Simply write the number.	128
Hexadecimal	Append an "0x" or "0X" suffix to the number. Append "h" or "H" at the end of the number.	0xA6, 0XA6 6Ah, 6AH

The beginning of a numeric constant should be a numeral.
For example, when 10 in decimal is written in hexadecimal with "H" appended at its end, append "0" at the beginning and write "0AH". If it is written as "AH", it is regarded as a symbol.

Caution Prefix notation (like 0xn...n) and suffix notation (n...nh) cannot be used together within one source program.
Specify the notation through the -base_number = (prefix | suffix) option.

(b) Character constants

A character constant consists of a single character enclosed by a pair of single quotation marks (' ') and indicates the value of the enclosed character.
The number of characters should be 1.
This is a 32-bit value holding the right-justified code for the specified character. When the upper bytes are empty, they are filled with 0.

Example

Character Constants	Evaluated Value
'A'	0x00000041
' ' (1 blank)	0x00000020

(c) Character string constants

A character string constant is a sequence of some characters shown in "(1) Character set" enclosed by a pair of quotation marks (" ") and indicates the characters themselves.

To include the double quote character in the string, write it twice in succession.

Example

Character string Constants	Evaluated Value
"ab"	0x6162
"A"	0x41
" " (1 blank)	0x20
""	None

(3) Symbol

A reference to a symbol is handled as a specification of the value defined for the symbol.
The symbols allowed in this assembler are classified into the following types.

- Name
  A symbol specified in the symbol field of a symbol definition directive. This type of symbol has a value.
  The range of a value is -2147483648 to 2147483647 (0x80000000 to 0x7FFFFFFF).
- Label
  A symbol written between the beginning of a line and a colon (:). This type of symbol has an address value.
  The range of an address value is 0 to 1048575 (0x00000 to 0xFFFFF).
- External reference name
  A symbol specified in the operand field of an external reference name definition directive to refer to the symbol defined in a module from another module. The address value for this symbol is set to 0 at assembly and it is determined at linkage.
  A symbol that is not defined in the module where the symbol is referenced is also regarded as an external reference name.
- Section name
  A symbol specified in the symbol field of a section definition directive.
  This symbol does not have a value.
- Macro name
  A symbol specified in the symbol field of a macro definition directive. It is used for reference to a macro.
  This symbol does not have a value.
- Macro formal parameter name
  A symbol specified in the operand field of a macro definition directive.
  This symbol does not have a value.

A symbol defined using a bit position specifier is called a bit symbol.
A reference to a symbol using a bit position specifier is called a bit reference to a symbol.

The symbol field is for symbols, which are names given to addresses and data objects. Symbols make programs easier to understand.

(a) Symbol types

Symbols that can be written in the symbol field are classified as shown below, depending on their purpose and how they are defined.

Symbol Type	Purpose	Definition Method
Label	Use this type when referring to the address of the label location. Note that the label appended to a directive is regarded as included in the section immediately before the directive	Write a symbol followed by a colon (:).
Name	Use this type when assigning numerical data or an address and referring to it as a symbol.	Write in the symbol field of a Symbol definition directive. Delimit the symbol field and mnemonic field by one or more spaces or tabs.
Section name	Use this type when referring to a symbol as input information for the optimizing linker.	Write in the symbol field of a section definition directive. Delimit the symbol field and mnemonic field by one or more spaces or tabs.
Macro name	Use to name macros in source programs.	Write in the symbol field of macro directive. Delimit the symbol field and mnemonic field by one or more spaces or tabs.

Multiple symbols cannot be written in a symbol field. In addition, only one symbol of any one of the above types can be defined in a line.

(b) Conventions of symbol description
Observe the following conventions when writing symbols.

- The characters which can be used in symbols are the alphanumeric characters and special characters (@, _, ., $).
  The first character in a symbol cannot be a digit (0 to 9) or $.
  To specify a symbol that includes periods for an operand of a bit manipulation instruction, enclose the symbol in double quotation marks.
  Example: set1 !"s.y.m".7
- The maximum number of characters for a symbol is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.
- Reserved words cannot be used as symbols.
  See "5.6 Reserved Words" for a list of reserved words.
- The same symbol cannot be defined more than once.
  However, a symbol defined with the .SET directive can be redefined with the .SET directive.
- When a label is written in a symbol field, the colon (:) must appear immediately after the label name.
  When using another type of symbol, insert a space or a tab to delimit the symbol from the mnemonic field.

Example Correct symbols

```
CODE01  .CSEG               ; "CODE01" is a section name.
VAR01   .EQU    0x10        ; "VAR01" is a name.
LAB01:  .DB2    0           ; "LAB01" is a label.
```

Example Incorrect symbols

```
1ABC    .EQU    0x3         ; The first character is a digit.s
LAB     MOV     1, r10      ; "LAB"is a label and must be separated from the
                            ; mnemonic field by a colon ( : ).
FLAG:   .EQU    0x10        ; The colon ( : ) is not needed for names.
```

Example A statement composed of a symbol only

```
ABCD:                       ; ABCD is defined as a label.
```

(c) Points to note about symbols
When writing an assembler generation symbol (see "5.7 Assembler Generated Symbols".), there is a possibility which becomes an error by a multi-definition, don't use an assembler generation symbol.
In addition, if a section name is not specified in a section definition directive, note that the assembler automatically generates a section name.

(4) Mnemonic field
Write instruction mnemonics, directives, and macro references in the mnemonic field.
If the instruction or directive or macro reference requires an operand or operands, the mnemonic field must be separated from the operand field with one or more blanks or tabs.
However, if the first operand begins with "#", "$","!", "[", or "(", the statement will be assembled properly even if nothing exists between the mnemonic field and the first operand field.

Example Correct mnemonics

```
MOV     A, #1
```

Example Incorrect mnemonics

```
MOVA, #1      ; There is no blank between the mnemonic and operand fields.
MO V  A, #1   ; The mnemonic field contains a blank.
MOVE  A, #1   ; This is an instruction that cannot be coded in the mnemonic field.
```

(5) Operand field
In the operand field, write operands (data) for the instructions, directives, or macro references that require them. Some instructions and directives require no operands, while others require two or more.
When you provide two or more operands, delimit them with a comma (,). Before and after a comma, any number of spaces or tabs can be inserted.

(6) Comment
Write a comment after a number sign (#) at the beginning of a line or after a semicolon (;) in the middle of a line
The comment field continues from the # or semicolon to the new line code at the end of the line, or to the EOF code of the file.
Comments make it easier to understand and maintain programs.
Comments are not processed by the assembler, and are output verbatim to assembly lists.
Characters that can be described in the comment field are those shown in "(1) Character set".

Example

```
# This is a comment
HERE:   MOV     A, #0x0F        ;This is a comment
;
;       BEGIN LOOP HERE
;
```

5.1.3 Expressions and operators

An expression is a Symbol[Note 1], constant (Numeric constants[Note 2], Character constants) , an operator combined with one of the above, or a combination of operators.

Note 1. Only a name, a label, or an external reference name can be used for a symbol specified as an element of an expression. For the SIZEOF and STARTOF operators, a section name can be specified.

Note 2. When a device file is read, SFR symbols and extended SFR symbols can be handled in an expression in the same way as constants.

Elements of an expression other than the operators are called terms, and are referred to as the 1st term, 2nd term, and so forth from left to right, in the order that they occur in the expression. The operators that can be used for a term are limited depending on the relocation attribute of the term.

The assembler supports the operators shown in "Table 5.4 Operator Types". Operators have priority levels, which determine when they are applied in the calculation. The priority order is shown in "Table 5.5 Operator Precedence Levels".
The order of calculation can be changed by enclosing terms and operators in parentheses "( )".

Table 5.4 Operator Types

Operator Type	Operators
Arithmetic operators	+, -, *, /, %, +sign, -sign
Bit logic operators	~, &, |, ^
Relational operators	==, !=, >, >=, <, <=
Logical operators	&&, ||
Shift operators	>>, <<
Byte separation operators	HIGH, LOW
2-byte separation operators	HIGHW, LOWW, MIRHW, MIRLW, SMRLW
Special operators	DATAPOS, BITPOS
Section operators	STARTOF, SIZEOF
Other operator	( )

The above operators can also be divided into unary operators, special unary operators and binary operators.

Unary operators	+sign, -sign, ~, HIGH, LOW, HIGHW, LOWW, MIRHW, MIRLW, SMRLW, DATA-POS, BITPOS, STARTOF, SIZEOF
Binary operators	+, -, *, /, %, &, |, ^, ==, =, >, >=, <, <=, &&, ||, >>, <<

Table 5.5 Operator Precedence Levels

Priority	Level	Operators
Higher	1	+sign, -sign, ~, HIGH, LOW, HIGHW, LOWW, MIRHW, MIRLW, SMRLW, DATA-POS, BITPOS, STARTOF, SIZEOF
	2	*, /, %, >>, <<
	3	+, -
	4	&, |, ^
	5	==, !=, >, >=, <, <=
Lower	6	&&, ||

Expressions are operated according to the following rules.

- The order of operation is determined by the priority level of the operators.
  When two operators have the same priority level, operation proceeds from left to right, except in the case of unary operators, where it proceeds from right to left.
- Sub-expressions in parentheses "( )" are operated before sub-expressions outside parentheses.
- Expressions are operated using 32-bit values.
  When the intermediate value of an expression or a constant in an expression exceeds 32 bits during evaluation or when the resultant value of the constant or the expression after evaluation exceeds 32 bits, only the lower-order 32 bits are valid. No error will be output in this case.
  Only for an expression specified as an operand for the .DB8 directive, each term is handled in 64 bits.
- Each term of an expression is handled as an unsigned integer, but in the following cases it is handled as a signed integer.
  Multiplication, division, division/multiplication, second term of logical shift
- If the divisor is 0, an error occurs.
- Negative values are represented as two's complement.
- Relocatable terms are evaluated as 0 at the time when the source is assembled (the evaluation value is determined at link time).

Table 5.6 Evaluation examples

Expression	Evaluation
5 + 8 - 6 * 2 / 4	10
5 + (8 - 6) * 2 / 4	6
(5 + 8 - 6)* 2 / 4	3
2 * (0x0F - (0x0B & (0x0A | 0x0F)))	8
2 * 0x0F - 0x0B & 0x0A | 0x0F	0x0F
HIGH(-1)	0xFF
HIGH(0x0FFFF)	0xFF
2 + 4 * 5	22
(2 + 3) * 4	20

Expression	Evaluation
10/4	2
0 - 1	0xFFFFFFFF
-1 > 1	1 (True)
EXT[Note] + 1	0

Note EXT: External reference names

5.1.4 Arithmetic operators

The following arithmetic operators are available.

Operator	Overview
+	Addition of values of first and second terms.
-	Subtraction of value of first and second terms.
*	Multiplacation of value of first and second terms.
/	Divides the value of the 1st term of an expression by the value of its 2nd term and returns the integer part of the result.
%	Obtains the remainder in the result of dividing the value of the 1st term of an expression by the value of its 2nd term.
+sign	Returns the value of the term as it is.
-sign	The term value 2 complement is sought.

```
+
```

Addition of values of first and second terms.

[Function]

Returns the sum of the values of the 1st and 2nd terms of an expression.

[Application example]

```
START:  BR      !!START + 6     ; (1)
```

(1) The BR instruction causes a jump to "address assigned to START plus 6", namely, to address "0x100 + 0x6 = 0x106" when START label is 0x100.

```
-
```

Subtraction of value of first and second terms.

[Function]

Returns the result of subtraction of the 2nd-term value from the 1st-term value.

[Application example]

```
BACK:   BR      !!BACK - 6      ; (1)
```

(1) The BR instruction causes a jump to "address assigned to BACK minus 6", namely, to address "0x100 - 0x6 = 0xFA" when BACK label is 0x100.

```
*
```

Multiplication of value of first and second terms.

[Function]

Returns the result of multiplication (product) between the values of the 1st and 2nd terms of an expression.

[Application example]

```
MOV     A, #2 * 3       ; (1)
```

(1) Execution of the MOV instruction loads a value of 6 in the A register.

```
/
```

Divides the value of the 1st term of an expression by the value of its 2nd term and returns the integer part of the result.

[Function]

Divides the value of the 1st term of an expression by the value of its 2nd term and returns the integer part of the result.
The decimal fraction part of the result will be truncated. If the divisor (2nd term) of a division operation is 0, an error occurs

[Application example]

```
MOV     A, #250 / 50    ; (1)
```

(1) Execution of the MOV instruction loads a value of 5 in the A register.

%

Obtains the remainder in the result of dividing the value of the 1st term of an expression by the value of its 2nd term.

[Function]

Obtains the remainder in the result of dividing the value of the 1st term of an expression by the value of its 2nd term.
An error occurs if the divisor (2nd term) is 0.

[Application example]

```
MOV     A, #256 % 50    ; (1)
```

(1) Execution of the MOV instruction loads a value of 6 in the A register.

+sign

Returns the value of the term as it is.

[Function]

Returns the value of the term of an expression without change.

-sign

The term value 2 complement is sought.

[Function]

Returns the value of the term of an expression by the two's complement.

5.1.5 Bit logic operators

The following bit logic operators are available.

Operator	Overview
~	Obtains the logical negation (NOT) by each bit.
&	Obtains the logical AND operation for each bit of the first and second term values.
|	Obtains the logical OR operation for each bit of the first and second term values.
^	Obtains the exclusive OR operation for each bit of the first and second term values.

```
~
```

Obtains the logical negation by each bit.

[Function]

Negates the value of the term of an expression on a bit-by-bit basis and returns the result.

[Application example]

```
MOV     A, #LOW(~3)     ; (1)
```

(1) Logical negation is performed on "0x3" as follows:
This operation loads "0x0F" in the A register.

NOT)	0000	0000	0000	0000	0000	0000	0000	0011
	1111	1111	1111	1111	1111	1111	1111	1100

```
&
```

Obtains the logical AND operation for each bit of the first and second term values.

[Function]

Performs an AND (logical product) operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

[Application example]

```
MOV     A, #0x6FA & 0x0F        ; (1)
```

(1) AND operation is performed between the two values "0x6FA" and "0x0F" as follows:
This operation loads "0xF0A" in the A register.

	0000	0000	0000	0000	0000	0110	1111	1010
AND)	0000	0000	0000	0000	0000	0000	0000	1111
	0000	0000	0000	0000	0000	0000	0000	1010

```
|
```

Obtains the logical OR operation for each bit of the first and second term values.

[Function]

Performs an OR (Logical sum) operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

[Application example]

```
MOV     A, #0x0A | 0b1101       ; (1)
```

(1) OR operation is performed between the two values "0x0A" and "0b1101" as follows:
This operation loads "0x0F" in the A register.

	0000	0000	0000	0000	0000	0000	0000	1010
OR)	0000	0000	0000	0000	0000	0000	0000	1101
	0000	0000	0000	0000	0000	0000	0000	1111

```
^
```

Obtains the exclusive OR operation for each bit of the first and second term values.

[Function]

Performs an Exclusive-OR operation between the value of the 1st term of an expression and the value of its 2nd term on a bit-by-bit basis and returns the result.

[Application example]

```
MOV     A, #0x9A ^ 0x9D         ; (1)
```

(1) XOR operation is performed between the two values "0x9A" and "0x9D" as follows:
This operation loads "0x07" in the A register.

	0000	0000	0000	0000	0000	0000	1001	1010
XOR)	0000	0000	0000	0000	0000	0000	1001	1101
	0000	0000	0000	0000	0000	0000	0000	0111

5.1.6 Relational operators

The following relational operators are available.

Operator	Overview
==	Compares whether values of first term and second term are equivalent.
!=	Compares whether values of first term and second term are not equivalent.
>	Compares whether value of first term is greater than value of the second.
>=	Compares whether value of first term is greater than or equivalent to the value of the second term.
<	Compares whether value of first term is smaller than value of the second.
<=	Compares whether value of first term is smaller than or equivalent to the value of the second term.

==

Compares whether values of first term and second term are equivalent.

[Function]

Returns 1 (True) if the value of the 1st term of an expression is equal to the value of its 2nd term, and 0 (False) if both values are not equal.

!=

Compares whether values of first term and second term are not equivalent.

[Function]

Returns 1 (True) if the value of the 1st term of an expression is not equal to the value of its 2nd term, and 0 (False) if both values are equal.

>

Compares whether value of first term is greater than value of the second.

[Function]

Returns 1(True) if the value of the 1st term of an expression is greater than the value of its 2nd term, and 0 (False) if the value of the 1st term is equal to or less than the value of the 2nd term.

>=

Compares whether value of first term is greater than or equivalent to the value of the second term.

[Function]

Returns 1 (True) if the value of the 1st term of an expression is greater than or equal to the value of its 2nd term, and 0 (False) if the value of the 1st term is less than the value of the 2nd term.

<

Compares whether value of first term is smaller than value of the second.

[Function]

Returns 1 (True) if the value of the 1st term of an expression is less than the value of its 2nd term, and 0 (False) if the value of the 1st term is equal to or greater than the value of the 2nd term.

<=

Compares whether value of first term is smaller than or equivalent to the value of the second term.

[Function]

Returns 1 (True) if the value of the 1st term of an expression is less than or equal to the value of its 2nd term, and 0 (False) if the value of the 1st term is greater than the value of the 2nd term.

5.1.7 Logical operators

The following logical operators are available.

Operator	Overview
&&	Calculates the logical product of the logical value of the first and second operands.
||	Calculates the logical sum of the logical value of the first and second operands.

&&

Calculates the logical product of the logical value of the first and second operands.

[Function]

Calculates the logical product of the logical value of the first and second operands.

||

Calculates the logical sum of the logical value of the first and second operands.

[Function]

Calculates the logical sum of the logical value of the first and second operands.

5.1.8 Shift operators

The following shift operators are available.

Operator	Overview
>>	Obtains only the right-shifted value of the first term which appears in the second term.
<<	Obtains only the left-shifted value of the first term which appears in the second term.

```
>>
```

Obtains only the right-shifted value of the first term which appears in the second term.

[Function]

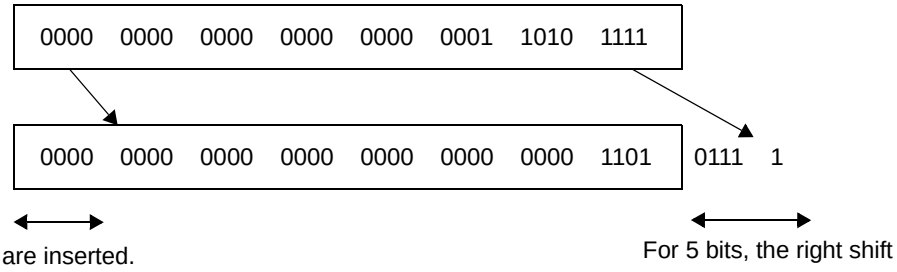
Returns a value obtained by shifting the value of the 1st term of an expression to the right the number of bits specified by the value of the 2nd term.
Zeros equivalent to the specified number of bits shifted move into the high-order bits.

If the number of shifted bits is 0, the value of the first term is returned as is. If the number of shifted bits exceeds 31, 0 is returned.

[Application example]

```
MOVW    AX, #0x01AF >> 5        ; (1)
```

(1) The value "0x01AF" is shifted 5 bits to the right, leaving the sign bit.
"0x000D" is forwarded to AX.



```
<<
```

Obtains only the left-shifted value of the first term which appears in the second term.

[Function]

Returns a value obtained by shifting the value of the 1st term of an expression to the left the number of bits specified by the value of the 2nd term.

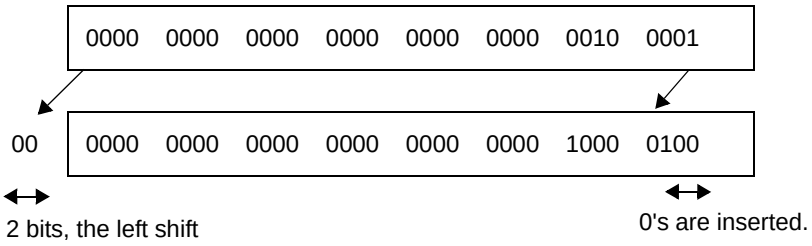
Zeros equivalent to the specified number of bits shifted move into the low-order bits.

If the number of shifted bits is 0, the value of the first term is returned as is. If the number of shifted bits exceeds 31, 0 is returned.

[Application example]

```
MOV     A, #0x21 << 2           ; (1)
```

(1) This operator shifts the value "0x21" to the left by 2 bits.
"0x84" is forwarded to A.



5.1.9 Byte separation operators

The following byte separation operators are available.

Operator	Overview
HIGH	Obtains the second byte from the least significant byte of a term
LOW	Returns the low-order 8-bit value of a term

HIGH

Obtains the second byte from the least significant byte of a term.

[Function]

Returns the value of bits 8 to 15 (the second byte from the least significant byte) among the 32 bits of a term.

[Application example]

```
MOV     A, #HIGH(0x1234)        ; (1)
```

(1) By executing a MOV instruction, this operator loads the high-order 8-bit value "0x12" of the expression "0x1234" to A register.

LOW

Returns the low-order 8-bit value of a term.

[Function]

Returns the value of the lower-order eight bits among the 32 bits of a term.

[Application example]

```
MOV     A, #LOW(0x1234)         ; (1)
```

(1) By executing a MOV instruction, this operator loads the low-order 8-bit value "0x34" of the expression "0x1234" to A register.

5.1.10 2-byte separation operators

The following 2-byte separation operators are available.

Operator	Overview
HIGHW	Returns the high-order 16-bit value of a term
LOWW	Returns the low-order 16-bit value of a term
MIRHW	Obtains the higher-order 16 bits of the corresponding address in the mirror destination area when the value of the specified term is in the mirror source area
MIRLW	Obtains the lower-order 16 bits of the corresponding address in the mirror destination area when the value of the specified term is in the mirror source area
SMRLW	Adds an offset to the mirror destination to the address of a symbol, adds an integer value to the obtained value, and then obtains the value of the lower-order 16 bits among the 32-bit value of the result

HIGHW

Returns the high-order 16-bit value of a term.

[Function]

Returns the value of the high-order 16 bits among the 32 bits of a term.

[Application example]

```
MOVW    AX, #HIGHW(0x12345678)          ; (1)
```

(1) By executing a MOVW instruction, this operator loads the high-order 16-bit value "0x1234" of the expression "0x12345678" to AX register.

LOWW

Returns the low-order 16-bit value of a term.

[Function]

Returns the value of the lower-order 16 bits among the 32 bits of a term.

[Application example]

```
MOVW    AX, #LOWW(0x12345678)          ; (1)
```

(1) By executing a MOVW instruction, this operator loads the low-order 16-bit value "0x5678" of the expression "0x12345678" to AX register.

MIRHW

Obtains the higher-order 16 bits of the corresponding address in the mirror destination area when the value of the specified term is in the mirror source area.

[Function]

When the value of the specified term is in the mirror source area, the higher-order 16 bits among the 32 bits of the corresponding address in the mirror destination area are returned.

When the value of the term is outside the mirror source area and the term is an absolute term (see "5.1.14 Restrictions on operations"), the same value as for HIGHW is returned. When it is a relocatable term, an error will occur at linkage.

[Application example]

```
MOVW    AX, #MIRHW(0x00001000)          ; (1)
```

(1) When the target expression (0x00001000) for operation is in the mirror source area, 0x00001000 is converted to the corresponding mirror destination address (0x000F9000 for an 8-bit CPU or 0x000F1000 for a 16-bit CPU) and the value of the higher-order 16 bits (0x000F) is loaded in the AX register by executing the MOVW instruction.

MIRLW

Obtains the lower-order 16 bits of the corresponding address in the mirror destination area when the value of the specified term is in the mirror source area.

[Function]

When the value of the specified term is in the mirror source area, the lower-order 16 bits among the 32 bits of the corresponding address in the mirror destination area are returned.

When the value of the term is outside the mirror source area and the term is an absolute term (see "5.1.14 Restrictions on operations"), the same value as for LOWW is returned. When it is a relocatable term, an error will occur at linkage.

[Application example]

```
MOVW    AX, #MIRLW(0x00001000)          ; (1)
```

(1) When the target expression (0x00001000) for operation is in the mirror source area, 0x00001000 is converted to the corresponding mirror destination address (0x000F9000 for an 8-bit CPU or 0x000F1000 for a 16-bit CPU) and the value of the lower-order 16 bits (0x9000 for an 8-bit CPU or 0x1000 for a 16-bit CPU) is loaded in the AX register by executing the MOVW instruction.

SMRLW

Adds an offset to the mirror destination to the address of a symbol, adds an integer value to the obtained value, and then obtains the value of the lower-order 16 bits among the 32-bit value of the result.

[Function]

When the specified term is an expression that adds an integer value to a relocatable symbol, only the symbol is checked instead of the value of the whole term. When the symbol is in the mirror source area, the offset to the mirror destination is added to the address of the symbol, the specified integer value is added to the obtained value, and then the value of the lower-order 16 bits among the 32-bit value of the result is returned.

When the relocatable symbol is outside the mirror source area, an error will occur at linkage.

When the term is an absolute term, the same value as for LOWW is returned. When the term only refers to a relocatable symbol, the same value as for MIRLW is returned.

Only a term that takes any one of the following forms after operation by the assembler is allowed. In other cases, errors will occur. (A is an absolute symbol, R is a relocatable symbol, and C is an integer constant in the following forms.)

This is also applicable to MIRLW and MIRHW.

- SMRLW(C) : Same operation as LOWW.
- SMRLW(A) : Same operation as LOWW.
- SMRLW(R) : R is checked as to whether it is in the mirror source area.
- SMRLW(R + C) : R is checked as to whether it is in the mirror source area.
- SMRLW(R - A + C) : R is checked as to whether it is in the mirror source area.

[Application example]

```
MOVW    AX, #SMRLW(GSYM + 0x1000)       ; (1)
```

GSYM is an external reference name.

(1) When the address of relocatable symbol GSYM is in the mirror source area, it is converted to the corresponding address in the mirror destination, 0x1000 is added to the obtained value, and then the value of the lower-order 16 bits of the result is loaded in the AX register.

5.1.11 Special operators

The following special operators are available.

Operator	Overview
DATAPOS	Obtains the first term of a bit symbol
BITPOS	Obtains the second term of a bit symbol

DATAPOS

Obtains the first term of a bit symbol.

[Function]

Obtains the first term of a bit symbol.

[Application example]

```
BITSYM  .EQU    0x0FE20.3
        MOVW    AX, #DATAPOS(BITSYM)    ; (1)
```

(1) Execution of the MOVW instruction loads a value of 0xFE20 in the AX register.

[Caution]

No bit term (see "5.1.15 Bit position specifier") can be specified in the operand field.

BITPOS

Obtains the second term of a bit symbol.

[Function]

Obtains the second term of a bit symbol.

[Application example]

```
BITSYM  .EQU    0x0FE20.3
        MOVW    AX, #BITPOS(BITSYM)     ; (1)
```

(1) Execution of the MOVW instruction loads a value of 3 in the AX register.

[Caution]

No bit term (see "5.1.15 Bit position specifier") can be specified in the operand field.

5.1.12 Section operators

The following Section operators are available.

Operator	Overview
STARTOF	Returns the start address of the term section after linking.
SIZEOF	Returns the size of the term section after linking.

STARTOF

Returns the start address of the section specified by the term after linking.

[Function]

Returns the start address of the term section after linking.

[Application example]

Allocates a 4-byte area, and initializes it with the start address of the default section (.text).

```
.DB4            STARTOF(.text)
```

To allocate a 4-byte area and initialize it with the start address of the user-defined section (user_text).

```
.DB4            STARTOF(user_text)
```

To use this operator in conjunction with SIZEOF:

```
.DB4            STARTOF(.data) + SIZEOF(.data)
```

[Caution]

- This operator can be specified in combination with SIZEOF by using the binary operator "+".
  Note, however, that it is not possible on the same line to write multiple instances of STARTOF and SIZEOF or include an expression other than STARTOF or SIZEOF. The following example will cause an error.

```
.DB4            STARTOF(.data) +2
```

- For an absolute section, write "_AT" + a section name with an address specified (see ".SECTION", ".CSEG", ".DSEG", and ".ORG").

```
.SECTION        EX, DATA_AT     0xF2000
.DB4            STARTOF(EX_ATF2000)
```

SIZEOF

Returns the size of the term section after linking.

[Function]

Returns the size of the term section after linking.

[Application example]

Allocates a 4-byte area, and initializes it with the size of the default section (.text).

```
.DB4    SIZEOF(.text)
```

To allocate a 4-byte area and initialize it with the size of the user-defined section (user_text).

```
.DB4            SIZEOF(user_text)
```

To use this operator in conjunction with STARTOF:

```
.DB4            STARTOF(.data) + SIZEOF(.data)
```

To specify EX_ATF2000 as the name for an absolute section.

```
.SECTION        EX, DATA_AT     0xF2000
.DB4            SIZEOF(EX_ATF2000)
```

[Caution]

- This operator can be specified in combination with SIZEOF by using the binary operator "+".
  Note, however, that it is not possible on the same line to write multiple instances of STARTOF and SIZEOF or include an expression other than STARTOF or SIZEOF.
- For an absolute section, write "_AT" + a section name with an address specified (see ".SECTION", ".CSEG", ".DSEG", and ".ORG").

5.1.13 Other operator

The following operators is also available.

Operator	Overview
( )	Prioritizes the calculation within ( ).

( )

Prioritizes the calculation within ( ).

[Function]

Causes an operation in parentheses to be performed prior to operations outside the parentheses.
This operator is used to change the order of precedence of other operators.
If parentheses are nested at multiple levels, the expression in the innermost parentheses will be calculated first.

[Application example]

```
MOV     A, #(4 + 3) * 2
```

Calculations are performed in the order of expressions (1), (2) and the value "14" is returned as a result.

If parentheses are not used,

Calculations are performed in the order (1), (2) shown above, and the value "10" is returned as a result.
See "Table 5.5 Operator Precedence Levels", for the order of precedence of operators.

5.1.14 Restrictions on operations

The operation of an expression is performed by connecting terms with operator(s). Elements that can be described as terms are constants, names and labels. Each term has a relocation attribute.

Depending on the types of relocation attribute inherent in each term, operators that can work on the term are limited. Therefore, when describing an expression it is important to pay attention to the relocation attribute of each term constituting the expression.

(1) Operators and relocation attributes

Each term constituting an expression has a relocation attribute.
If terms are categorized by relocation attribute, they can be divided into 2 types: absolute terms and relocatable terms.
The following table shows the types of relocation attributes and their properties, and also the corresponding terms.

Table 5.7 Relocation Attribute Types

Type	Property	Corresponding Elements
Absolute term	Term that is a value or constant determined at assembly time	- Constants - Names for which constants are defined
Relocatable term	Term with a value that is not determined at assembly time	- Labels - Names for which labels are defined - Labels defined with .EXTERN directive - Names defined with .EXTBIT directive - Symbols not defined in the module

The following tables categorize combinations of operators and terms which can be used in expressions by relocation attribute.

Table 5.8 Combinations of Operators and Terms by Relocation Attribute

Operator Type	Relocation Attribute of Term			
	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
+ X	A	A	R	R
- X	A	A	-	-
~ X	A	A	-	-
HIGH X	A	A	R[Note 1]	R[Note 1]
LOW X	A	A	R[Note 1]	R[Note 1]
HIGHW X	A	A	R[Note 1]	R[Note 1]
LOWW X	A	A	R[Note 1]	R[Note 1]
MIRHW X	A	A	R[Note 2]	R[Note 2]
MIRLW X	A	A	R[Note 2]	R[Note 2]
SMRLW X	A	A	R[Note 2]	R[Note 2]
DATAPOS X.Y	-	-	-	-
BITPOS X.Y	-	-	-	-
DATAPOS X	A	A	-	-
BITPOS X	A	A	-	-

Operator Type	Relocation Attribute of Term			
	X:ABS Y:ABS	X:ABS Y:REL	X:REL Y:ABS	X:REL Y:REL
X + Y	A	R	R	-
X - Y	A	-	R	R
X * Y	A	-	-	-
X / Y	A	-	-	-
X % Y	A	-	-	-
X >> Y	A	-	-	-
X << Y	A	-	-	-
X & Y	A	-	-	-
X | Y	A	-	-	-
X ^ Y	A	-	-	-
X == Y	A	-	-	-
X != Y	A	-	-	-
X > Y	A	-	-	-
X >= Y	A	-	-	-
X < Y	A	-	-	-
X <= Y	A	-	-	-
X && Y	A	-	-	-
X || Y	A	-	-	-

ABS : Absolute term
REL : Relocatable term
A : Result is absolute term
R : Result is relocatable term
- : Operation not possible

Note 1. Operation is possible when X is not relocatable terms operated on by MIRHW, MIRLW, SMRLW, or DATAPOS.

Note 2. Operation is possible when X is not relocatable terms operated on by HIGH, LOW, HIGHW, LOWW, MIRHW, MIRLW, SMRLW, or DATAPOS.

(2) Nesting of operators
The HIGH, HIGHW, LOW, and LOWW operators can be specified in a nested manner.

(3) Absolute expression and relative expression
Expressions are classified into absolute and relative expressions, which are handled separately.

(a) Absolute expression
An expression indicating a constant is called an "absolute expression". An absolute expression can be used when an operand is specified for an instruction or when a value etc. is specified for a directive. An absolute expression usually consists of a constant or symbol. The following format is treated as an absolute expression.

<1> Constant expression
If a reference to a previously defined symbol is specified, it is assumed that the constant of the value defined for the symbol has been specified. Therefore, a defined symbol reference can be used in a constant expression.
However, a symbol that is not defined or whose value is not determined when the symbol is referenced is not handled as a constant expression

Example

```
SYM1    .EQU    0x10        ;Define symbol SYM1
        MOV     A, #SYM1    ;SYM1, already defined, is treated as a constant
                             expression.
```

<2> Symbol
The expressions related to symbols are the following ("±" is either "+" or "-").

- Symbol
- Symbol ± constant expression
- Symbol - symbol
- Symbol - symbol ± constant expression

A "symbol" here means a symbol that is an absolute term (a name for which a constant is defined somewhere in the module) but that is not defined or whose value is not determined yet when it is referenced.. If a reference to a previously defined symbol is specified, it is assumed that the "constant" of the value defined for the symbol has been specified.

Example

```
        MOV     A, #SYM1        --SYM1 is an undefined symbol at this point
SYM1    .EQU    0x10            --Defines SYM1
```

(b) Relative expressions
An expression indicating an offset from a specific address[Note 1] is called a "relative expression". A relative expression is used to specify an operand by an instruction or to specify a value by data definition directive. A relative expression usually consists of a symbol (label and external reference name).
The following format[Note 2] is treated as an relative expression("±" is either "+" or "-").

Note 1. This address is determined when the optimizing linker is executed. Therefore, the value of this offset may also be determined when the optimizing linker is executed.

Note 2. It can regard an expression in the format of "-symbol + label reference", as being an expression in the format of "label reference - symbol," but it cannot regard an expression in the format of "label reference - (+symbol)" as being an expression in the format of "label reference - symbol". Therefore, use parentheses "( )" only in constant expressions.

- Symbol
- Symbol ± constant expression
- Symbol - symbol[Note]
- Symbol - symbol ± constant expression[Note]

Note A label cannot be used as a symbol after "-", except for subtraction between labels.

When any of the specified symbols is a relocatable term, the expression is handled as a relative expression. Here is an example of a relative expression.

Example

```
SIZE    .EQU    0x10
        MOV     A, #label1
        MOV     A, #label1 + 0x10
        MOV     A, #label2 ? SIZE
        MOV     A, #label2 ? SIZE + 0x10
```

5.1.15 Bit position specifier

Bit access becomes possible via use of the (.) bit position specifier.

(1) Description Format

```
address.bit-position
```

(2) Function
The first term specifies an address, and the second term specifies a bit position. This makes it possible to access a specific bit.

(3) Explanation

- The term obtained through a bit position specifier is called a bit term, which has a bit value.
- A bit term cannot be used as a term in an expression.
- The bit position specifier is not affected by the precedence order of operators. The left side is recognized as the first term and the right side is recognized as the second term.
- The following restrictions apply to the first term:
  - A bit term can be used as an operand for an instruction that handles bit data (such as MOV1) (For details, see the user's manual of the device).
  - If the first term is an absolute expression, the area must be 0x00000 to 0xFFFFF.
  - External reference names can be specified.
- The following restrictions apply to the second term:
  - The value of the absolute expression must be in the range from 0 to 7. When this range is exceeded, an error occurs.
  - External reference names cannot be specified.

(4) Operations and relocation attributes
The following table shows combinations of terms 1 and 2 by relocation attribute.

Terms combination X:	ABS	ABS	REL	REL
Terms combination Y:	ABS	REL	ABS	REL
X.Y	A	-	R	-

ABS : Absolute term
REL : Relocatable term
A : Result is absolute term
R : Result is relocatable term
- : Operation not possible

(5) Example

```
MOV1    CY, 0xFFE20.3
AND1    CY, A.5
CLR1    P1.2
SET1    1 + 0xFFE30.3   ;Equals 0xFFE31.3 ((1 + 0xFFE30) is the first term and 3
                        ;is the second term)
SET1    0xFFE40.4 + 2   ;Equals 0xFFE40.6 (0xFFE40 is the first term and (4 + 2)
                        ;is the second term)
```

5.1.16 Operand characteristics

Instructions and directives requiring one or more operands differ in the size and address range of the required operand values of the operands.

For example, the function of the instruction "MOV r, #byte" is to transfer the value indicated by "byte" to register "r". Because the register is an 8-bit register, the data size of "byte" must be 8 bits or less.

An assembly error will occur at the statement "MOV R0, 0x100", because the value of the second operand (0x100) cannot be expressed with 8 bits.

Therefore, it is necessary to bear the following points in mind when describing operands.

- Whether the size and address range are suitable for an operand of that instruction (numeric value, name, label)

(1) Operand value sizes and address ranges

There are conditions that limit the size and address ranges of numeric values, names and labels used as instruction operands.

For instructions, the size and address range of operands are limited by the operand representation. For directives, they are limited by the directive type.

These limiting conditions are as follows.

Table 5.9 Instruction Operand Value Ranges

Operand Representation	Value Range	
byte	8-bit value : 0x00 to 0xFF	
word	word [B] word [C] word [BC]	- Numeric constants 0x0000 to 0xFFFF - Labels 0xF0000 to 0xFFFFF[Note 1] When a label is in the mirror source area[Note 2], the corresponding address in the mirror destination area[Note 2] is masked to be a 16-bit value and this value should be within the above range
	ES : word [B] ES : word [C] ES : word [BC]	- Numeric constants 0x0000 to 0xFFFF - Labels 0x00000 to 0xFFFFF The ES value is not checked for its valid range
	Other than the above	16-bit value : 0x0000 to 0xFFFF
saddr	0xFFE20 to 0xFFF1F[Note 3] Note that the saddr area range depends on the device	
saddrp	0xFFE20 to 0xFFF1F even numbe[Note 3] Note that the saddr area range depends on the device	
sfr	0xFFF20 to 0xFFFFF : Special function register symbols (SFR symbols[Note 4]), numeric constants, and symbols	
sfrp	0xFFF20 to 0xFFFFE : Special function register symbols (SFR symbols[Note 4]), numeric constants, and symbols(even values only)	
addr5	0x00080 to 0x000BF (CALLT table area, even values only)	

Operand Representation	Value Range	
addr16	!addr16 (BR, CALL instructions)	0x0000 to 0xFFFF (The range in which numeric constants and symbols can be specified is the same)
	!addr16[Note 5] (Other than BR, CALL instructions)	- Numeric constants[Note 6] 0x0000 to 0xFFFF - Labels[Note 6] 0xF0000 to 0xFFFFF[Note 1] When a label is in the mirror source area[Note 2], the corresponding address in the mirror destination area[Note 2] is masked to be a 16-bit value and this value should be within the above rang
	ES:!addr16	- Numeric constants[Note 6] 0x0000 to 0xFFFF - Labels[Note 6] 0x00000 to 0xFFFFF The ES value is not checked for its valid range
	!addr16.bit	- When addr16 is a numeric constant 0x0000 to 0xFFFF - When addr16 or addr16.bit is a label 0xF0000 to 0xFFFFF[Note 1] When a label is in the mirror source area[Note 2], the corresponding address in the mirror destination area[Note 2] is masked to be a 16-bit value and this value should be within the above rang
	ES : !addr16.bit	- When addr16 is a numeric constant 0x0000 to 0xFFFF - When addr16 or addr16.bit is a label 0x00000 to 0xFFFFF The ES value is not checked for its valid range
addr20	$addr20	0x00000 to 0xFFFFF, and when a branch destination is in the range (-0x80) to (+0x7F) from the next address after a branch instruction
	$!addr20	0x000000 to 0xFFFFFF, and when a branch destination is in the range (-0x8000) to (+0x7FFF) from the next address after a branch instruction
	!!addr20	0x00000 to 0xFFFFF
bit	3-bit value : 0 to 7	
RBn	n:2-bit value : 0 to 3	

Note 1. The mirror destination area and the internal RAM area are the valid ranges, and the actual ranges of these areas are determined by referring to the device file. When the device file is not referred to, the valid range is 0xF0000 to 0xFFFFF.

Note 2. The address range of the mirror source area differs depending to the device. For details, see the user's manual of the device.

Note 3. The saddr area is determined by referring to the device file. When the device file is not referred to, the valid range is 0xFFE20 to 0xFFF1F.

Note 4. The address range for SFR symbols is determined by referring to the device file. When the device file is not referenced, SFR symbols must not be used. The address range for SFR symbols is 0xFFF00 to 0xFFFFF, but the address range from 0xFFF00 to 0xFFF1F is regarded as saddr even if an SFR symbol is used.

Note 5. When an SFR symbol or an extended SFR (2ndSFR) symbol is used as an operand, "!SFR" or "!2ndSFR" can be written as "!addr16" and a code for "!addr16" is generated.
The address range for extended SFR symbols is also determined by referring to the device file.

Note 6. Only an even value is allowed for an operand in a 16-bit instruction (16-bit data transfer instruction or 16-bit operation instruction).

The range differs between numeric constants and labels for the following reason.
When a code is generated for operand "word" or "addr16", the range of the values that can be output is 0x0000 to 0xFFFF. Therefore, when a numeric constant is specified as an operand, it is checked for this range. However, when a label is specified as an operand, the range is determined as follows considering the meaning of each value.

(a) word
The actual location to be accessed is 0xF0000 to 0xFFFFF in the based addressing. Therefore, a label is checked for this address range.

(b) addr16
The actual location to be accessed is 0xF0000 to 0xFFFFF except for the BR and CALL instructions. Therefore, a label is checked for this address range.

Table 5.10 Value ranges of Directive Operands

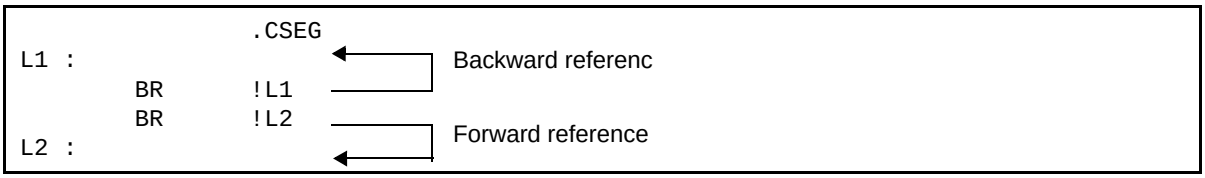
Directive Type	Directive	Value Range
Section definition	.ORG	0x00000 to 0xFFFFF
	.OFFSET	0x00000 to 0xFFFFF
Symbol definition	.EQU	0x00000000 to 0xFFFFFFFF For a bit symbol, the range is as follows: Address value : 0x00000 to 0xFFFFF Bit value : 0 to 7
	.SET	0x00000000 to 0xFFFFFFFF
Data definition/Area reservation	.DB	Initial value setting: 0x00 to 0xFF
	.DB2	Initial value setting: 0x0000 to 0xFFFF
	.DB4	Initial value setting: 0x00000000 to 0xFFFFFFFF
	.DB8	Initial value setting: 0x00000000 00000000 to 0xFFFFFFFF FFFFFFFF
	.DS	Size setting : 0x00000 to 0xFFFFF
	.ALIGN	Alignment condition value : 2 or a greater even number less than 2^{31}

(2) Sizes of operands required by instructions
Instructions can be classified into machine instructions and directives. When they require immediate data or symbols, the size of the required operand differs according to the instruction or directive. An error occurs when source code specifies data that is larger than the required operand.
Evaluation of an expression is done in 32 bits, both during calculation and for the calculation result. Therefore, even an overflow value is handled in 32 bits.
However, when a relocatable symbol is specified as an operand, its value cannot be determined by the assembler. In this case, the linker determines the value and checks its range.

(3) Symbol attribute required by instructions
Among the instructions that allow a symbol to be specified as an operand, the attribute (absolute, relocatable, or external reference) of symbols that can be specified differ depending on the instruction.
Reference direction for symbols can be backward reference or forward reference.

- Backward reference : A symbol referenced as an operand, which is defined in a line above (before) the name or label
- Forward reference : A symbol referenced as an operand, which is defined in a line below (after) the name or label

<Example>



The following shows the attributes of symbols that can be specified as operands for machine-language instructions.

Table 5.11 Properties of Described Symbols as Operands

	Relocation Attributes	Attributes		Relocatable Note 1		SFR Reserved Words Note 2	
	Reference Pattern	Backward	Forward	Backward	Forward	SFR	2ndSFR Note 3
Description Format	byte	OK	OK	OK	OK	-	-
	word	OK	OK	OK	OK	-	-
	saddr	OK	OK	OK	OK	OK Note 4,5	-
	saddrp	OK	OK	OK	OK	OK Note 4,6	-
	sfr	-	-	-	-	OK Note 4,7	-
	sfrp	-	-	-	-	OK Note 4,8	-
	addr20	OK	OK	OK	OK	OK	OK
	addr16	OK	OK	OK	OK	OK Note 9	OK Note 9
	addr5	OK	OK	OK	OK	-	-
	bit	OK	-	-	-	-	-

Forward : Forward reference
Backward : Backward reference
OK : Description possible
- : An error

Note 1. When a relocatable symbol is used, the optimizing linker determines its value and checks its range.

Note 2. The defined symbol specifying sfr or sfrp (sfr area where saddr and sfr are not overlapped) as an operand of .EQU directive is only referenced backward. Forward reference is prohibited.

Note 3. 2nd SFR : 2nd Special Function Register

Note 4. If an SFR symbols in the saddr area has been described for an instruction in which a combination of sfr/sfrp changed from saddr/saddrp exists in the operand combination, a code is output as saddr/saddrp.

Note 5. 8-bit SFR in saddr area

Note 6. 16-bit SFR in saddr area

Note 7. 8-bit SFR

Note 8. 16-bit SFR

Note 9. !SFR, !2ndSFR, and SFR can be specified only for operand !addr16 of instructions other than BR and CALL.

Table 5.12 Properties of Described Symbols as Operands of Directives

	Relocation Attributes	Attributes		Relocatable Note 1	
	Reference Pattern	Backward	Forward	Backward	Forward
Directive	.ORG	OK Note 2	-	-	-
	.OFFSET	OK Note 2	-	-	-
	.EQU	OK Note 2	-	-	-
	.SET	OK Note 2	-	-	-
	.DB	OK	OK	OK	OK
	.DB2	OK	OK	OK	OK
	.DB4	OK	OK	OK	OK
	.DB8	OK Note 2	-	-	-
	.DS	OK Note 2	-	-	-
	.ALIGN	OK Note 2	-	-	-

Forward : Forward reference
Backward : Backward reference
OK : Description possible
- : Description impossible

Note 1. When a relocatable symbol is used, the optimizing linker determines its value and checks its range.

Note 2. Only an absolute expression can be described.

5.2 Directives

This section explains the directives.
Directives are instructions that direct all types of instructions necessary for the assembler.

5.2.1 Outline

Instructions are translated into machine language as a result of assembling, but directives are not converted into machine language in principle.
Directives contain the following functions mainly:

- To facilitate description of source programs
- To initialize memory and reserve memory areas
- To provide the information required for assemblers and optimizing linkers to perform their intended processing

The following table shows the types of directives.

Table 5.13 List of Directives

Type	Directives
Section definition directives	.SECTION, .CSEG, .DSEG, .ORG, .OFFSET
Symbol definition directives	.EQU, .SET
Data definition/Area reservation directives	.DB, .DB2, .DB4, .DB8, .DS, .ALIGN
External definition/External reference directives	.PUBLIC, .EXTERN, .EXTBIT
Compiler output directives	.LINE, .STACK, ._LINE_TOP, ._LINE_END, .VECTOR
Macro directives	.MACRO, .LOCAL, .REPT, .IRP, .EXITM, .EXITMA, .ENDM
Branch directives	.Bcond

The following sections explain the details of each directive.
In the description format of each directive, "[ ]" indicates that the parameter in square brackets may be omitted from specification, and "..." indicates the repetition of description in the same format.

5.2.2 Section definition directives

A "section definition directive" is a directive that indicates the start or end of a section.
Sections are the unit of allocation in the optimizing linker.

Example

```
.SECTION    SecA, TEXT
    :
.SECTION    SecB, DATA
    :
.SECTION    SecC, BSS
    :
```

Two sections with relocation attributes SBSS and SBSS_BIT may have the same section name. Two sections with relocation attributes BSS and BSS_BIT may have the same section name.

Regarding other relocation attributes, two sections with the same section name must have the same relocation attribute. Consequently, multiple sections with differing relocation attributes cannot be given the same section name. If two sections with the same section name have different relocation attributes, an error will occur. The contents that can be written in a section depend on the relocation attribute of the section. For the details of description, see "Table 5.15 Relocation Attributes".

Sections can be broken up. In other words, sections in a single source program file with the same relocation attribute and section name will be processed as a single continuous section in the assembler.

If a section is broken into separate modules (assembly source files), then they are linked by the optimizing linker.
by the optimizing linker.

The start address can be specified for a section. The section with the start address specified is an absolute section.

The following section definition directives are available.

Table 5.14 Section Definition Directives

Directive	Overview
.SECTION	Indicates to the assembler the start of a section
.CSEG	Indicates to the assembler the starting of a code section
.DSEG	Indicates to the assembler the start of a data section
.BSEG	Indicates to the assembler the start of a bit section
.ORG	Indicates to the assembler the start of a section at an absolute address
.OFFSET	Specifies an offset from the first address of a section

.SECTION

Indicate to the assembler the start of section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.SECTION	*section-name*, *relocation-attribute* [, ALIGN=*absolute-expressions*] [, COMDAT=*signature-name*]	[; *comment*]

ALIGN can be specified in V1.10 or later.
COMDAT can be specified in V1.12 or later.

[Function]

- The .SECTION directive indicates to the assembler the start of a section (no separation of code and data).

[Description]

- This directive defines a program or data that has a coherent set of functions in a program.
  This directive is valid until another section definition directive appears.
- When an instruction that outputs a label or an object code is used at the beginning of a source program before this directive appears, a relocatable code section is generated as a default section.
  In this case, the section name will be ".text", and the relocation attribute is set to "TEXT".
- .SECTION directive can specify the start address of a section by specifying AT, DATA_AT, BSS_AT, or BIT_AT as the relocation attribute in the operand field.
  The section start address can also be specified through the .ORG directive.
  In this case, the section name will be "the section name specified in the operand field" + "_AT" + "specified address (hexadecimal notation in uppercase letters without prefix (0x or 0X) or suffix (h or H)".
- The following shows the relocation attributes that can be specified.
  If an attribute that is not listed below is used, an error will occur.

Table 5.15 Relocation Attributes

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Align-ment Condi-tion[Note 1]
CALLT0	CALLT0	.callt0	Allocates a section between addresses 0x00080 and 0x000BF in the code flash area[Note 2] with the start address set to an even address.	2
TEXT	TEXT	.text	Allocates a section between addresses 0x000C0 and 0x0FFFF in the code flash area[Note 2].	1
TEXTF	TEXTF	.textf	Allocates a section between addresses 0x000C0 and 0xEFFFF in the code flash area[Note 2].	1
TEXTF_UNIT64KP	TEXTF_UNIT64KP	.textf_unit64kp	Allocates a section with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2[Note 4]

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Align-ment Condi-tion[Note 1]
CONST	CONST	.const	Allocates a section in the mirror source area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
CONSTF	CONSTF	.constf	Allocates a section in the code flash area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
SDATA	SDATA	.sdata	Allocates a section for data having initial values in the saddr area[Note 2] with the start address set to an even address.	2
SBSS	SBSS	.sbss	Allocates a section for data having no initial values in the saddr area[Note 2] with the start address set to an even address.[Note 8]	2
SBSS_BIT	SBSS_BIT	.sbss_bit	Allocates a section for bits having no initial values in the saddr area[Note 2] with the start address set to an even address. The optimizing linker links this section in byte units and assumes the relocation attribute as SBSS.[Note 8 Note 9]	2
DATA	DATA	.data	Allocates a section for data having initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
BSS	BSS	.bss	Allocates a section for data having no initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].[Note 8]	2
BSS_BIT	BSS_BIT	.bss_bit	Allocates a section for bits having no initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3]. The optimizing linker links this section in byte units and assumes the relocation attribute as BSS.[Note 8 Note 9]	2
DATAF	DATAF	.dataf	Allocates a section for data having initial values with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
BSSF	BSSF	.bssf	Allocates a section for data having no initial values with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
AT∆ad-dress	AT absolute- expres-sion[Note 5]	None	Allocates a section at a specified address.	1 (fixed)

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Align-ment Condi-tion[Note 1]
DATA_AT∆ address	DATA_AT absolute-expression[Note 5]	None	Allocates a section for data having initial values at a specified address.	1 (fixed)
BSS_AT∆address	BSS_AT absolute-expression[Note 5]	None	Allocates a section for data having no initial values at a specified address.[Note 8]	1 (fixed)
BIT_AT∆address	BIT_AT absolute-expression[Note 5]	None	Allocates a section for bits having no initial values at a specified address. The optimizing linker links this section in byte units and assumes the relocation attribute as BSS_AT.[Note 8 Note 11]	1 (fixed)
OPT_BYTE	OPT_BYTE	.option_byte[Note 6]	This attribute is dedicated to the user option byte setting and on-chip debugging setting[Note 7].	1 (fixed)
SECUR_ID	SECUR_ID	.security_id[Note 6]	This attribute is dedicated to the security ID setting-[Note 7]. Machine-language instructions cannot be written in this section.	1 (fixed)
FLASH_SECUR_ID	FLASH_SECUR_ID	.flash_security_id[Note 6]	This attribute is dedicated to the flash programmer security ID setting[Note 7]. Machine-language instructions cannot be written in this section.	1 (fixed)

Note 1. The alignment condition can be modified through the .ALIGN directive.

Note 2. For the code flash area, mirror area, RAM area, and saddr area, see the user's manual of the device. For the RAM area, note that only the on-chip RAM allocated to an address range from 0xF0000 to 0xFFFFF is supported.

Note 3. Allocation beyond a boundary of 64 Kbytes - 1 is prohibited by default.

Note 4. To guarantee correct access to 16-bit data, the alignment condition value is set to "2".

Note 5. If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.

Note 6. This is a special section and the section name must not be changed (the name is fixed).

Note 7. For addresses where the option byte, on-chip debugging, and security ID settings are allocated, see the user's manual of the device.

Note 8. The two sections with the same name and different relocation attributes SBSS and SBSS_BIT, relocation attributes BSS and BSS_BIT, or relocation attributes BSS_AT and BIT_AT are processed as a single consecutive section in the assembler.

Note 9. The section is output to the object file with the relocation attribute of SBSS, and the optimizing linker allocates the section with the relocation attribute of SBSS.

Note 10. The section is output to the object file with the relocation attribute of BSS, and the optimizing linker allocates the section with the relocation attribute of BSS.

Note 11. The section is output to the object file with the relocation attribute of BSS_AT, and the optimizing linker allocates the section with the relocation attribute of BSS_AT.

- The section name setting cannot be omitted.
- The relocation attribute setting cannot be omitted.
- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

- You can change the default alignment condition by specifying the ALIGN parameter. For the .ALIGN directive, specifying a larger value than that specified in the ALIGN parameter results in an error. [V1.10.00 or later]
- When the COMDAT parameter is specified, only one section selected from among the sections having the same name and same signature is linked. [V1.12.00 or later]

[Example]

To define section ".text" having the TEXT attribute.

```
.SECTION        .text , TEXT
NOP
```

To define section ".data" having the DATA attribute.

```
.SECTION        "data", DATA
.DB2            0x1
```

To define section "EX" having the DATA_AT attribute with address 0xf2000 specified.
The section name will be set to "EX_ATF2000".

```
.SECTION        EX, DATA_AT     0xf2000
.DS             4
```

.CSEG

Indicate to the assembler the start of a code section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*section-name*]	.CSEG	[*relocation-attribute*]	[; *comment*]

[Function]

- The .CSEG directive indicates to the assembler the start of a code section.
- All instructions described following the .CSEG directive belong to the code section until it comes across a section definition directives.

[Description]

- This directive defines a portion that has a coherent set of functions in a program.
  This directive is valid until another section definition directive appears.
- When an instruction that outputs a label or an object code is used at the beginning of a source program before this directive appears, a relocatable code section is generated as a default section.
  In this case, the section name will be ".text", and the relocation attribute is set to "TEXT".
- .CSEG directive can specify the start address of a section by specifying AT in the operand field..
  The section start address can also be specified through the .ORG directive.
  In this case, the section name will be "the specified section name" + "_AT" + "specified address (hexadecimal notation in uppercase letters without prefix (0x or 0X) or suffix (h or H)".
- The following shows the relocation attributes that can be specified through .CSEG.

Table 5.16 Relocation Attributes of .CSEG

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Alignment Condition[Note 1]
CALLT0	CALLT0	.callt0	Allocates a section between addresses 0x00080 and 0x000BF in the code flash area[Note 2] with the start address set to an even address.	2
TEXT	TEXT	.text	Allocates a section between addresses 0x000C0 and 0x0FFFF in the code flash area[Note 2] with the start address set to an even address.	1
TEXTF	TEXTF	.textf	Allocates a section between addresses 0x000C0 and 0xEFFFF in the code flash area[Note 2] with the start address set to an even address.	1
TEXTF_UNIT64KP	TEXTF_UNIT64KP	.textf_unit64kp	Allocates a section with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2[Note 4]
CONST	CONST	.const	Allocates a section in the mirror source area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Alignment Condition[Note 1]
CONSTF	CONSTF	.constf	Allocates a section in the code flash area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
AT∆address	AT absolute-expression[Note 5]	None	Allocates a section at a specified address.	1 (fixed)
OPT_BYTE	OPT_BYTE	.option_byte[Note 6]	This attribute is dedicated to the user option byte setting and on-chip debugging setting[Note 7].	1 (fixed)
SECUR_ID	SECUR_ID	.security_id[Note 6]	This attribute is dedicated to the security ID setting[Note 7]. Machine-language instructions cannot be written in this section.	1 (fixed)
FLASH_SECUR_ID	FLASH_SECUR_ID	.flash_security_id[Note 6]	This attribute is dedicated to the flash programmer security ID setting[Note 7]. Machine-language instructions cannot be written in this section.	1 (fixed)

Note 1. The alignment condition can be modified through the .ALIGN directive.

Note 2. For the code flash area, mirror area, RAM area, and saddr area, see the user's manual of the device. For the RAM area, note that only the on-chip RAM allocated to an address range from 0xF0000 to 0xFFFFF is supported.

Note 3. Allocation beyond a boundary of 64 Kbytes - 1 is prohibited by default.

Note 4. To guarantee correct access to 16-bit data, the alignment condition value is set to "2".

Note 5. If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.

Note 6. This is a special section and the section name must not be changed (the name is fixed).

Note 7. For addresses where the option byte, on-chip debugging, and security ID settings are allocated, see the user's manual of the device.

- When a section definition does not include a section name, the assembler gives a separate default section name for each relocation attribute.
The following shows the section names given by the assembler.

Relocation Attribute	Section Name
CALLT0	.callt
TEXT	.text
TEXTF	.textf
TEXTF_UNIT64KP	.textf_unit64kp
CONST	.const
CONSTF	.constf
AT∆address	.text_AT start-address
OPT_BYTE	.option_byte[Note]
SECUR_ID	.security_id[Note]
FLASH_SECUR_ID	.flash_security_id[Note]

Note This is a special section and the section name must not be changed (the name is fixed).

A section having one of the above names has the corresponding relocation attribute shown above and no different relocation attribute can be assigned.

- When a section definition does not include a relocation attribute, relocation attribute "TEXT" is assumed.
- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

[Example]

To define section ".text" having the TEXT attribute.

```
.text   .CSEG   TEXT
        NOP
```

To define section ".unit" having the TEXTF_UNIT64KP attribute.

```
unit    .CSEG   TEXTF_UNIT64KP
        MOV     A, !LABEL
```

To define section "EX" having the AT attribute with address 0x00200 specified.
The section name will be set to "EX_AT200".

```
EX      .CSEG   AT 0x00200
        .DS     2
```

To define a section for the option byte setting.

```
.CSEG   OPT_BYTE
.DB     0xFF
```

.DSEG

Indicate to the assembler the start of a data section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*section-name*]	.DSEG	[*relocation-attribute*]	[; *comment*]

[Function]

- The .DSEG directive indicates to the assembler the start of a data section.
- A memory following the .DSEG directive belongs to the data section until it comes across a section definition directives.

[Description]

- This directive defines a portion that defines data in a program.
  This directive is valid until another section definition directive appears.
- DSEG directive can specify the start address of a section by specifying DATA_AT nad BSS_AT in the operand field.
  The section start address can also be specified through the .ORG directive.
  In this case, the section name will be "the specified section name" + "_AT" + "specified address (hexadecimal notation in uppercase letters without prefix (0x or 0X) or suffix (h or H)".
- The following shows the relocation attributes that can be specified through .DSEG.

Table 5.17 Relocation Attributes of .DSEG

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Alignment Condition[Note 1]
SDATA	SDATA	.sdata	Allocates a section for data having initial values in the saddr area[Note 2] with the start address set to an even address.	2
SBSS	SBSS	.sbss	Allocates a section for data having no initial values in the saddr area[Note 2] with the start address set to an even address.	2
DATA	DATA	.data	Allocates a section for data having initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
BSS	BSS	.bss	Allocates a section for data having no initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 2] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
DATAF	DATAF	.dataf	Allocates a section for data having initial values with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Align-ment Condi-tion[Note 1]
BSSF	BSSF	.bssf	Allocates a section for data having no initial values with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 3].	2
DATA_AT∆ address	DATA_AT absolute-expression[Note 4]	None	Allocates a section for data having initial values at a specified address.	1 (fixed)
BSS_AT∆address	BSS_AT absolute-expression[Note 4]	None	Allocates a section for data having no initial values at a specified address.	1 (fixed)

Note 1. The alignment condition can be modified through the .ALIGN directive.

Note 2. For the code flash area, mirror area, RAM area, and saddr area, see the user's manual of the device. For the RAM area, note that only the on-chip RAM allocated to an address range from 0xF0000 to 0xFFFFF is supported.

Note 3. Allocation beyond a boundary of 64 Kbytes - 1 is prohibited by default.

Note 4. If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.

- A directive that specifies initial values cannot be written in a section definition for data having no initial values. If described, an error is output.
- When a section definition does not include a section name, the assembler gives a separate default section name for each relocation attribute.
  The following shows the section names given by the assembler

Relocation Attribute	Section Name
SDATA	.sdata
SBSS	.sbss
DATA	.data
BSS	.bss
DATAF	.dataf
BSSF	.bssf
DATA_AT∆address	.data_AT start-address
BSS_AT∆address	.bss_AT start-address

A section having one of the above names has the corresponding relocation attribute shown above and no different relocation attribute can be assigned.

- When a section definition does not include a relocation attribute, relocation attribute "DATA" is assumed.
- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

[Example]

To define section ".data" having the DATA attribute.

```
.data   .DSEG   DATA
        .DS     4
```

To define section "_S" having the SDATA attribute.

```
_S      .DSEG   SDATA
        .DS     4
```

To define section "EX" having the DATA_AT attribute with address 0xff000 specified.
The section name will be set to "EX_ATFF000".

```
EX      .DSEG   DATA_AT 0xff000
        .DS     2
```

.BSEG

Indicate to the assembler the start of a bit section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*section-name*]	.BSEG	[*relocation-attribute*]	[; *comment*]

[Function]

- The .BSEG directive indicates to the assembler the start of a bit section.
- A memory following the .BSEG directive belongs to the bit section until it comes across a section definition directives.

[Description]

- This directive defines a portion that defines bit data in a program.
  This directive is valid until another section definition directive appears.
- This directive can specify the start address of a section by specifying BIT_AT in the operand field.
  In this case, the section name will be "the specified section name" + "_AT" + "specified address (hexadecimal notation in uppercase letters without prefix (0x or 0X) or suffix (h or H)".
- The following shows the relocation attributes that can be specified through .BSEG.

Table 5.18 Relocation Attributes of .BSEG

Relocation Attribute	Description Format	Default Section Name	Explanation	Default Value of Align-ment Condi-tion
SBSS_BIT	SBSS_BIT	.sbss_bit	Allocates a section for bits having no initial values in the saddr area[Note 1] with the start address set to an even address. The optimizing linker links this section in byte units and assumes the relocation attribute as SBSS.[Note 4 Note 5]	2
BSS_BIT	BSS_BIT	.bss_bit	Allocates a section for bits having no initial values between addresses 0xF0000 and 0xFFFFF in the RAM area[Note 1] with the start address set to an even address so that it does not extend across a boundary of 64 Kbytes - 1[Note 2]. The optimizing linker links this section in byte units and assumes the relocation attribute as BSS.[Note 4 Note 6]	2
BIT_AT∆address	BIT_AT absolute-expression[Note 3]	None	Allocates a section for bits having no initial values at a specified address. The optimizing linker links this section in byte units and assumes the relocation attribute as BSS_AT.[Note 4 Note 7]	1 (fixed)

Note 1. For the code flash area, mirror area, RAM area, and saddr area, see the user's manual of the device. For the RAM area, note that only the on-chip RAM allocated to an address range from 0xF0000 to 0xFFFFF is supported.

Note 2. Allocation beyond a boundary of 64 Kbytes - 1 is prohibited by default.

Note 3. If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.

Note 4. The two sections with the same name and different relocation attributes SBSS and SBSS_BIT, relocation attributes BSS and BSS_BIT, or relocation attributes BSS_AT and BIT_AT are processed as a single consecutive section in the assembler.

Note 5. The section is output to the object file with the relocation attribute of SBSS, and the optimizing linker allocates the section with the relocation attribute of SBSS.

Note 6. The section is output to the object file with the relocation attribute of BSS, and the optimizing linker allocates the section with the relocation attribute of BSS.

Note 7. The section is output to the object file with the relocation attribute of BSS_AT, and the optimizing linker allocates the section with the relocation attribute of BSS_AT.

- The instructions that can be written in a section defined by this directive are the .DBIT, .EQU, .SET, .PUBLIC, .EXTBIT, and .EXTERN directives and macro calls.
  An error will occur when other machine instructions or directives are written.
- When a section definition does not include a section name, the assembler gives a separate default section name for each relocation attribute.
  The following shows the section names given by the assembler.

Relocation Attribute	Section Name
SBSS_BIT	.sbss_bit
BSS_BIT	.bss_bit
BIT_AT∆address	.bit_AT start-address

A section having one of the above names has the corresponding relocation attribute shown above and no different relocation attribute can be assigned.

- When a section definition does not include a relocation attribute, relocation attribute "SBSS_BIT" is assumed.
- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

[Example]

To define section ".sbss_bit" having the SBSS_BIT attribute.

```
.sbss_bit  .BSEG  SBSS_BIT
sym01      .DBIT
```

To define section "_B" having the BSS_BIT attribute.

```
_B      .BSEG   BSS_BIT
        .DBIT
sym02   .DBIT
```

To define section "EX" having the BIT_AT attribute with address 0xffe20 specified.
The section name will be set to "EX_ATFFE20".

```
EX      .BSEG   BIT_AT 0xffe20
sym03   .DBIT
        .DBIT
```

.ORG

Indicate the start of a section at an absolute address to the assembler.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.ORG	*absolute-expression*	[; *comment*]

[Function]

- Indicate the start of a section at an absolute address to the assembler.

[Description]

- The range from the .ORG directive to the line with the next section definition directive (.CSEG, .DSEG, .SECTION or .ORG) is regarded as a section where the code is placed at absolute addresses.
- The name of an absolute addressing section will be "the name of the section for which the .ORG directive is written (excluding the "_AT" and the subsequent characters for an absolute addressing section)" + "_AT" + "specified address (hexadecimal notation in uppercase letters without prefix (0x or 0X) or suffix (h or H)". The relocation attribute will be the same as that of the section for which .ORG is written.
- If .ORG is written prior to a section definition directive at the beginning of a file of source code, the name of the section will be ".text.AT" + "specified address" and the relocation attribute will be "TEXT".
- The operand value is in accordance with "(a) Absolute expression". If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.
- The overall definition of a single section may contain multiple .ORG directives. However, if a section definition already exists for the section name specified through this directive or the section address specified through this directive is in an address range where another absolute addressing section within the same module is already allocated, an error will occur.

[Example]

If .ORG is written immediately after a section definition directive, the section is only generated from the absolute address.

```
        .SECTION    My_text, text
        .ORG        0x12            ;My_text.AT12 is allocated to address 0x12
LAB1:   MOV         A, !LABEL
        .ORG        0x30            ;My_text.AT30 is allocated to address 0x30
        MOV         A,!LABEL
```

If the .ORG directive does not immediately follow the section definition directive, only the range of code from the .ORG directive is a section starting at the given absolute address.

```
.SECTION    "My_text", text
NOP                             ;Allocated in My_text
.ORG        0x50
MOV         A,!LABEL            ;Allocated in My_text_AT50
```

If .ORG is written in an absolute addressing section, the section name will be "the absolute addressing section name before "_AT" " + "_AT" + "specified address".

```
.SECTION    My_text, AT     0x20
NOP                                 ;Allocated in My_text_AT20
.ORG        0x50
MOV         A,!LABEL                ;Allocated in My_text_AT50
```

.OFFSET

Specifies an offset from the first address of a section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.OFFSET	*absolute-expression*	[; *comment*]

[Function]

- The .OFFSET directive specifies an offset from the first address of a section that holds instruction code or data for the lines following the .OFFSET directive.
- After the .ORG directive, it is valid until the next section definition directive.

[Description]

- If .OFFSET is written prior to any section definition directive at the beginning of a source program, the name of the section will be ".text" and the relocation attribute will be "TEXT".
- The operand value is in accordance with "(a) Absolute expression". If the specified absolute expression is illegal or its value is outside the range from 0x00000 to 0xFFFFF, an error will occur.
- The overall definition of a single section may contain multiple .ORG directives. Note, however, that an error occurs when the specified value is smaller than that for a preceding .OFFSET directive.
- The initial value for the area between the address of this directive line and the offset address specified by this directive is "0x0".
- The .OFFSET directive cannot be used when the relocation attribute for the target section includes BSS. If the directive is specified in this case, an error will occur.

[Example]

```
.SECTION    My_data, text
.OFFSET     0x12
MOV         A, B            ; The offset is 0x12
```

5.2.3 Symbol definition directives

Symbol definition directives specify symbols for the data that is used when writing to source modules. With these, the data value specifications are made clear and the details of the source module are easier to understand.

Symbol definition directives indicate the symbols of values used in the source module to the assembler.

The following symbol definition directives are available.

Table 5.19 Symbol Definition Directives

Directive	Overview
.EQU	Defines a name having an absolute-expression value
.SET	Defines a name having an absolute-expression value

.EQU

Defines a name having an absolute-expression value.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
name	.EQU	*absolute-expression*[*.bit-position*]	[; *comment*]
name	.EQU	PSW[*.bit-position*]	[; *comment*]

[Function]

Defines a name having a absolute-expression value specified by the operand field.

[Use]

- You can use this directive to define names for numerical data that can be used instead of the actual numbers in the operands of machine-language instructions and directives in source code.
- We recommend defining frequently used numerical values as names. Even if a given value in the source program is to be changed, you will only need to change the value corresponding to the name.

[Description]

- The .SET directive may be described anywhere in a source program.
- Symbols that have already been defined by using .EQU cannot be redefined.
- The name generated by the .EQU directive can be externally defined by the .PUBLIC directive.
- The following values can be specified as the operand.
  - Absolute expression
    The absolute expression should be in accordance with "(a) Absolute expression".
  - PSW
    When PSW is specified, the value is 0xFFFFA.

  An illegal operand will cause an error.
- A value between 0x00000000 and 0xFFFFFFFF can be specified as the operand, and a value between 0 and 7 can be specified as the bit position.
- Relocatable terms cannot be specified in the operand field. In addition, operators that generate a relocatable term as a result of calculation are not allowed in the operand field.

[Example]

To specify a constant expression.

```
SYM1    .EQU    10
        MOV     A, #SYM1
```

To specify a symbol.

```
SYM2    .EQU    0xFFE20
BSYM1   .EQU    SYM2.1
        SET1    BSYM1
```

To specify PSW.

```
SYM3    .EQU    PSW
        MOV     SYM3, #10
```

To specify a SFR symbol.

```
SYM4    .EQU    P0
        MOV     SYM4, #2
```

.SET

Defines a name having an absolute-expression value.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
name	.SET	*absolute-expression*	[; *comment*]

[Function]

Defines a name having a absolute-expression value specified by the operand field.

[Use]

- You can use this directive to define names for numerical data that can be used instead of the actual numbers in the operands of machine-language instructions and directives in source code.
- We recommend defining frequently used numerical values as names. Even if a given value in the source program is to be changed, you will only need to change the value corresponding to the name.

[Description]

- The .SET directive may be described anywhere in a source program.
- Each name is a redefinable name.
- The name generated by the .SET directive cannot be externally defined by the .PUBLIC directive.
- The operand value is in accordance with "(a) Absolute expression". An illegal value will lead to an error.
- A value between 0x00000000 and 0xFFFFFFFF can be specified as an absolute expression in the operand field.
- This directive differs from the .EQU directive in that a symbol with a bit position specification cannot be defined.
- Operators that generate a relocatable term as a result of calculation are not allowed in the operand field.

[Example]

To specify a constant expression.

```
SYM1    .SET    10
        MOV     A, #SYM1
```

To specify SFR symbols.

```
SYM2    .SET    P0
        MOV     SYM2, #2
```

5.2.4 Data definition/Area reservation directives

The data definition directive defines the constant data used by the program.
The defined data value is generated as object code.
The area reservation directive allocates the area for memory used by the program.
The following data definition and partitioning directives are available.

Table 5.20 Data Definition/Area Reservation Directives

Directive	Overview
.DB	Initialization of byte area
.DB2	Initialization of 2-byte area
.DB4	Initialization of 4-byte area
.DB8	Initialization of 8-byte area
.DS	Allocates the memory area of the number of bytes specified by operand
.DBIT	Allocates a bit area of one bit
.ALIGN	Aligns the value of the location counter

.DB

Initialization of byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.DB	{*expression*|*"Character string constants"*}[, ...]	[; *comment*]

[Function]

- The .DB directive tells the assembler to initialize a byte area.

[Description]

- The .DB directive tells the assembler to initialize byte area..

  (a) Expression
  The value of an expression must be 1 byte of data. Therefore, the value of the operand must be in the range of 0x0 to 0xFF. The assembler checks the lower-order 24 bits of the result of an operation and outputs an error message if the value is not within the range from 0x0 to 0xFF.
  An expression may include a relocatable symbol or reference to an external name.

  (b) Character string constants
  If an operand is surrounded by corresponding double quotes ("), then it is assumed to be a string constant.
  If a character string constant is specified as the operand, a required number of bytes are allocated.

- If the relocation attribute of the section containing the .DB directive is "BSS", then an error is output because initial values cannot be specified.

[Example]

```
        .DSEG   DATA
LABEL:  .DB     10      ; 1-byte area is initialized by 10
        .DB     "ABC"   ; 3-byte area is initialized by
                        ; character string "ABC" (0x41, 0x42, 0x43)
```

.DB2

Initialization of 2-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.DB2	*expression*[, ...]	[; *comment*]

[Function]

- The .DB2 directive tells the assembler to initialize 2-byte area.

[Description]

- The .DB2 directive tells the assembler to initialize 2-byte area.
  The value of an expression must be 2 bytes of data. Therefore, the value of the operand must be in the range of 0x0000 to 0xFFFF. The assembler checks the lower-order 24 bits of the result of an operation and outputs an error message if the value is not within the range from 0x0000 to 0xFFFF.
  An expression may include a relocatable symbol or reference to an external name.
- Character string constants cannot be specified in the operand field.
- If the relocation attribute of the section containing the .DB2 directive is "BSS", then an error is output because initial values cannot be specified.

[Example]

```
        .DSEG   DATA
LABEL:  .DB2    0x1234  ; 2-byte area is initialized by 0x34, 0x12
```

.DB4

Initialization of 4-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.DB4	*expression*[, ...]	[; *comment*]

[Function]

- The .DB4 directive tells the assembler to initialize 4-byte area.

[Description]

- The .DB4 directive tells the assembler to initialize 4-byte area.
  The value of an expression must be 4-byte data. Therefore, the value of the operand must be in the range of 0x0 to 0xFFFFFFFF. If the value exceeds 4-byte data, the assembler will use only lower 4-byte value as valid data.
  An expression that includes a relocatable symbol or external reference name may be described.
- Character string constants cannot be specified in the operand field.
- If the relocation attribute of the section containing the .DB4 directive is "BSS", then an error is output because initial values cannot be specified.

[Example]

```
        .DSEG   DATA
LABEL:  .DB4    0x12345678  ; 4-byte area is initialized by 0x78, 0x56, 0x34, 0x12
```

.DB8

Initialization of 8-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.DB8	*absolute-expression*[, ...]	[; *comment*]

[Function]

- The .DB8 directive tells the assembler to initialize 8-byte area.

[Description]

- The .DB8 directive tells the assembler to initialize 8-byte area..
  The value of an absolute-expression must be 8-byte data. Therefore, the value of the operand must be in the range of 0x00000000 00000000 to 0xFFFFFFFF FFFFFFFF. If the value exceeds 8-byte data, the assembler will use only lower 8-byte value as valid data.
- An expression that conforms to "(a) Absolute expression" can be specified as the operand.
  However, the .DB8 directive handles each term in 64 bits. Therefore, a name (32 bits) defined through the .EQU directive cannot be used as a negative value in the .DB8 directive.
- Character string constants cannot be specified in the operand field.
- If the relocation attribute of the section is "BSS", then an error is output because the .DB8 directive cannot be described.

[Example]

```
        .DSEG   DATA
LABEL:  .DB8    0x1234567890ABCDEF  ; 8-byte area is initialized by
                                    ; 0xEF, 0xCD, 0xAB, 0x90, 0x78, 0x56, 0x34, 0x12
```

.DS

Allocates the memory area of the number of bytes specified by operand.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.DS	*absolute-expression*	[; *comment*]

[Function]

- Allocates an area for the number of bytes specified in the operand.

[Description]

- For a section for data having no initial values, allocates an area for the number of bytes specified in the operand. For other sections, allocates an area for the number of bytes specified in the operand and initializes it with 0.
  Note, however, that no area will be allocated if the specified number of bytes is 0.
- An expression that conforms to "(a) Absolute expression" can be specified as the area size.
- If the specified size is illegal or exceeds the range from 0x00000 to 0xFFFFF, an error will be output.
- When a label is specified, it is defined as a symbol whose value is the start address of the allocated area.

[Example]

```
        .DSEG   DATA
AREA1:  .DS     4       ; 4-byte area is allocated and initialized with 0
        .DSEG   BSS
AREA2:  .DS     8       ; 8-byte area is allocated
```

.DBIT

Allocates a bit area of one bit.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*name*]	.DBIT		[; *comment*]

[Function]

- Allocates a bit area of one bit.

[Description]

- This directive can be written in only a bit section.
- This directive is used to allocate a bit area of one bit in a bit section.
- The contents of the allocated bit area are undefined.
- When a name is written in the symbol field, it will be defined as a bit symbol which has an address and bit location as its value.

[Example]

```
        .BSEG
BSYM1   .DBIT   ; Bit area of one bit is allocated
        .DBIT   ; Bit area of one bit is allocated
BSYM2   .DBIT   ; Bit area of one bit is allocated
        .CSEG
        SET1    BSYM1
        CLR1    BSYM2
```

.ALIGN

Aligns the value of the location counter.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.ALIGN	*alignment-condition*	[; *comment*]

[Function]

- Aligns the value of the location counter.

[Description]

- Aligns the value of the location counter for the current section, specified by the previously specified section definition directive under the alignment condition specified by the operand. The area created to align the location counter value is filled with 0 (However, except the section with the relocation attribute includes BSS).
- Specify an even number of 2 or more, but less than 2^{31}, as the alignment condition. Otherwise, the CC-RL outputs the error message.
- This directive just aligns the value of the location counter for the current section within the module in which this directive is used. It does not align the address after sections are allocated through the optimizing linker.

[Example]

```
        .CSEG   TEXT
        .DS     1       ;OFFSET 0x0
        .ALIGN  2       ;OFFSET 0x2 ;1-byte padding
LABEL:                  ;OFFSET 0x2 ;If the beginning of the section is placed at an
                        ;odd address during allocation through the optimizing linker,
                        ;the alignment condition value is not set to 2.
                        ;(When the start address is 0x1, this code is placed at 0x3.)
```

```
        .SECTION  D1, DATA
        .DB       1
        .ALIGN    4
        .DB       2
        .ALIGN    6
        .DB       3
        ; Since the alignment condition of a section is the least common multiple
        ; of the alignment conditions included in the section, the alignment condition
        ; of section D1 is 12.
```

5.2.5 External definition/External reference directives

External definition, external reference directives clarify associations when referring to symbols defined by other modules.

This is thought to be in cases when one program is written that divides module 1 and module 2. In cases when you want to refer to a symbol defined in module 2 in module 1, there is nothing declared in either module and and so the symbol cannot be used. Due to this, there is a need to display "I want to use" or "I don't want to use" in respective modules.

An "I want to refer to a symbol defined in another module" external reference declaration is made in module 1. At the same time, a "This symbol may be referred to by other symbols" external definition declaration is made in module 2.

This symbol can only begin to be referred to after both external reference and external definition declarations in effect.

External definition, external reference directives are used to to form this relationship and the following instructions are available.

Table 5.21 External Definition/External Reference Directives

Directive	Overview
.PUBLIC	Declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module
.EXTERN	Declares to the optimizing linker that a symbol in another module is to be referenced in this module
.EXTBIT	Declares to the optimizing linker that a bit symbol in another module is to be referenced in this module
.WEAK [V1.11 or later]	Declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module

.PUBLIC

Declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.PUBLIC	*symbol-name*	[; *comment*]

[Function]

- The .PUBLIC directive declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module.

[Use]

- When defining a symbol to be referenced from another module, the .PUBLIC directive must be used to declare the symbol as an external definition.

[Description]

- When the symbol(s) to be described in the operand field isn't defined within the same module, a warning is output. The symbol name is output in the error message.
  When the symbol(s) isn't defined in any module, it will cause an error during linking.
- The following symbols cannot be used as the operand of the .PUBLIC directive:
  (a) Symbol defined with the .SET directive
  (b) Section name
  (c) Macro name

[Example]

```
        .PUBLIC  PSYM01
        .PUBLIC  PSYM02
PSYM01: .DB      0x10
PSYM02:
        MOV      A, #0x4D
```

.EXTERN

Declares to the optimizing linker that a symbol in another module is to be referenced in this module.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.EXTERN	*symbol-name*	[; *comment*]

[Function]

- The .EXTERN directive declares to the optimizing linker that a symbol in another module is to be referenced in this module.

[Description]

- The symbol specified in the operand field can be referenced in the current module.
- The following cannot be written as an operand.
  - (a) Symbol defined with the .SET directive
  - (b) Section name
  - (c) Macro name
- No error is output even if a symbol declared with the .EXTERN directive is not referenced in the given module.
- The .EXTERN directive may be described anywhere in a source program.

[Example]

Referring program

```
        .EXTERN  PSYM01
        .EXTERN  PSYM02
        MOV      A, ES:!PSYM01
        BR       !PSYM02
```

Defining program

```
        .PUBLIC  PSYM01
        .PUBLIC  PSYM02
PSYM01: .DB      0x10
PSYM02: MOV      A, #0x4D
```

.EXTBIT

Declares to the optimizing linker that a bit symbol in another module is to be referenced in this module.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.EXTBIT	*symbol-name*	[; *comment*]

[Function]

- The .EXTBIT directive declares to the optimizing linker that a bit symbol in another module is to be referenced in this module.

[Description]

- The bit symbol specified in the operand field can be referenced in the current module.
- The following cannot be written as an operand.
  (a) Symbol defined with the .SET directive
  (b) Section name
  (c) Macro name
- No error is output even if a bit symbol declared with the .EXTBIT directive is not referenced in the given module.
- The .EXTBIT directive may be described anywhere in a source program.

[Example]

```
      .EXTBIT  EBIT01
      .EXTBIT  EBIT02
      MOV1     EBIT01, CY
      AND1     CY, EBIT02
```

.WEAK [V1.11 or later]

Declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.WEAK	*symbol-name*	[; *comment*]

[Function]

- The .WEAK directive declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module.

[Description]

- The .WEAK directive declares to the optimizing linker that the symbol described in the operand field is a symbol to be referenced from another module.
- The differences between the .WEAK directive and the .PUBLIC directive are as follows:
  - If symbols with the same name exist in different modules, specifying the .PUBLIC directive for each of these symbols will cause an error during linking.
  - If symbols with the same name exist in different modules, specifying the .PUBLIC directive for one of these symbols and specifying the .WEAK directive for others does not cause an error. In this case, the module specified by the .PUBLIC directive will be linked.
- The .WEAK directive may be described anywhere in a source program.
- The following cannot be written as an operand:
  (a) Symbol defined with the .SET directive
  (b) Section name
  (c) Macro name

5.2.6 Compiler output directives

Compiler output directives inform the assembler of information output by the compiler, such as compiler debugging information.

The following compiler output directives are available.

Table 5.22 Compiler Output Directives

Directive	Overview
.LINE	Line-number information from the C source program
.STACK	Defines the amount of stack usage for a symbol
._LINE_TOP	Information specified by the compiler #pragma inline_asm statement
._LINE_END	Information specified by the compiler #pragma inline_asm statement
.VECTOR	Generates a interrupt vector table
.ALIAS [V1.11.00 or later]	Sets information about a symbol
.TYPE [V1.11.00 or later]	Sets information about a symbol

.LINE

Line-number information from the C source program.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.LINE	["*file-name*",] *line-number*	[; *comment*]

[Function]

- The .LINE directive is compiler debugging information.

[Description]

- Modifies the line numbers and filenames referenced during debugging.
- The line numbers and filenames in the source program are not updated between the first .LINE directive and the next one.
- If the filename is omitted, then only the line number is changed.

[Caution]

- The information handled by the .LINE directive is the line-number information of the C source program that the compiler outputs. The user must not use this directive.

.STACK

Defines the amount of stack usage for a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.STACK	*symbol-name=absolute-expression*	[; *comment*]

[Function]

- The .STACK directive is compiler debugging information.

[Description]

- This defines the amount of stack usage for a symbol.
- The amount of stack usage for a symbol can only be defined once, and subsequent definitions are ignored.
- The amount of stack usage can only be a multiple of 2 within the range of 0x0000 to 0xFFFE.
- The information handled by the .STACK directive is the function information of the C source program that the compiler outputs.

._LINE_TOP

Information specified by the compiler #pragma inline_asm statement.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	._LINE_TOP	inline_asm	[; *comment*]

[Function]

- The ._LINE_TOP directive is the information specified by the compiler #pragma inline_asm statement.

[Description]

- This is the #pragma inline_asm statement information of the C source program that the compiler outputs.
- The ._LINE_TOP directive indicates the start of the instructions for a function which has been specified as inline_asm.

[Caution]

- Assembler control instructions are not usable in assembly code for functions specified as inline_asm. In addition, only the directives listed below are usable. Specifying any other directive will lead to an error.
  - data definition/area reservation directives (.DB/.DB2/.DB4/.DB8/.DS)
  - macro directives (.MACRO/.IRP/.REPT/.LOCAL/.ENDM)
  - externally defined directive (.PUBLIC) [V1.04 or later]
- In the .PUBLIC directive in the function specified with inline_asm, only the labels defined in the function specified with inline_asm can be used. Using any other labels will lead to errors.
- The information handled by the ._LINE_TOP directive is the line-number information of the C source program that the compiler outputs. The user must not use this directive.

._LINE_END

Information specified by the compiler #pragma inline_asm statement.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	._LINE_END	inline_asm	[; *comment*]

[Function]

- The ._LINE_END directive is the information specified by the compiler #pragma inline_asm statement.

[Description]

- This is the #pragma inline_asm statement information of the C source program that the compiler outputs.
- The ._LINE_END directive indicates the end of the instructions for a function which has been specified as inline_asm.

[Caution]

- Assembler control instructions are not usable in assembly code for functions specified as inline_asm. In addition, only the directives listed below are usable. Specifying any other directive will lead to an error.
  - data definition/area reservation directives (.DB/.DB2/.DB4/.DB8/.DS)
  - macro directives (.MACRO/.IRP/.REPT/.LOCAL/.ENDMl)
  - externally defined directive (.PUBLIC) [V1.04 or later]
- In the .PUBLIC directive in the function specified with inline_asm, only the labels defined in the function specified with inline_asm can be used. Using any other labels will lead to errors.
- The information handled by the ._LINE_END directive is the line-number information of the C source program that the compiler outputs. The user must not use this directive.

.VECTOR

The .VECTOR directive generates an interrupt vector table.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
symbol-name	.VECTOR	*Vector-table-allocation-address*	[; *comment*]

[Function]

- The .VECTOR directive generates an interrupt vector table.

[Description]

- Specifies a vector table allocation address for an interrupt function.
- The symbol name should be the label of the destination of a branch to be executed when an interrupt occurs.
- An even address between 0x00000 and 0x0007E can be specified as the vector table allocation address. If a value outside this range is specified, an error will occur.
- Multiple addresses can be specified for a function.
- The information handled by the .VECTOR directive is the vector table information output by the compiler for #pragma interrupt specified in the C source program. When this directive is included in the code (#pragma interrupt is specified in the C source program), this directive should be used instead of a section definition directive to define a vector table in the assembly source program such as in the startup routine.
- When both the vector tables specified through the .VECTOR directive and those specified through the section definition directive are used together, the optimizing linker outputs an error.

[Example]

To specify multiple addresses (0x00008 and 0x0000a) for interrupt function intfunc.

```
_intfunc        .VECTOR 0x00008
_intfunc        .VECTOR 0x0000a
.SECTION        .text, TEXT
_intfunc:
```

To specify multiple addresses (0x00020, 0x00022, 0x00024, 0x00026 and 0x00028) for interrupt function intfunc2.

```
ADDR            .SET    0x00020
                .REPT   5
_intfunc2       .VECTOR ADDR
ADDR            .SET    ADDR + 2
                .ENDM
.SECTION        .text, TEXT
_intfunc2:
```

.ALIAS [V1.11.00 or later]

The .ALIAS directive sets information about a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.ALIAS	*alias, symbol-name*	[; *comment*]

[Description]

- This directive generates an alias of the symbol.
- The generated alias can be specified in the .PUBLIC directive.
- If a label name that does not exist in any module is specified for the symbol name, an error will occur.
- This directive must be described after the definition of the symbol specified by the symbol name on the operand.

[Example]

To allow only the alias symA to be referenced from outside the module, specify as follows.

```
symB:
.PUBLIC symA
.ALIAS symA, symB
```

```
SECTION=
SYMBOL ADDR SIZE INFO COUNTS OPT

 symA
   00000120 0 none ,g 0
 symB
   00000120 0 none ,l 0
```

.TYPE [V1.11.00 or later]

The .TYPE directive sets information about a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.TYPE	*symbol-name, symbol-type, size-specification*	[; *comment*]

[Description]

- This directive specifies the type and size information of the symbol.
- "FUNCTION" or "OBJECT" can be specified for the symbol type.
- If a label name that does not exist in any file is specified for the symbol name, an error will occur.

[Example]

```
.TYPE symA, FUNCTION, 8
symA:
```

The set information can be viewed in the link map.

```
SYMBOL ADDR SIZE INFO COUNTS OPT
  symA
   00000000 8 func ,l 0
```

5.2.7 Macro directives

When describing a source it is inefficient to have to describe for each series of high usage frequency instruction groups. This is also the source of increased errors.

Via macro directives, using macro functions it becomes unnecessary to describe many times to the same kind of instruction group series, and coding efficiency can be improved.

Macro basic functions are in substitution of a series of statements.

The following macro directives are available.

Table 5.23 Macro Directives

Directive	Overview
.MACRO	Defines a set of statements written between the .MACRO and .ENDM directives as a macro having the name specified in the symbol field
.LOCAL	Defines the specified symbol name as a local symbol that is valid only within the macro body where it is defined
.REPT	Tells the assembler to repeatedly expand a series of statements described between .REPT directive and the .ENDM directive the number of times equivalent to the value of the absolute-expression specified in the operand field.
.IRP	Tells the assembler to repeatedly expand a series of statements described between .IRP directive and the .ENDM directive the number of times equivalent to the number of actual parameters while replacing the formal parameter with the actual parameters (from the left, the order) specified in the operand field.
.EXITM	This directive skips the repetitive assembly of the .IRP and .REPT directives enclosing this directive at the innermost position.
.EXITMA	This directive skips the repetitive assembly of the irp and .REPT directives enclosing this directive at the outermost position.
.ENDM	Instructs the assembler to terminate the execution of a series of statements defined as the functions of the macro.

.MACRO

Defines a set of statements written between the .MACRO and .ENDM directives as a macro having the name specified in the symbol field.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
name	.MACRO : *Macro body* :	[*formal-parameter*[, ...]]	[; *comment*]

[Function]

- Defines a set of statements (called "a macro body") written between the .MACRO and .ENDM directives as a macro having the name specified in the symbol field.
  When this macro name is referenced (called "a macro call"), it is handled as the macro body corresponding to the macro name written at that location (called "a macro expansion").

[Description]

- If there is no .ENDM directive corresponding to a .MACRO directive within the same file, the CCRL outputs a message.
  For example, if only .MACRO or .ENDM exists in an include file, an error will occur.
- The maximum number of formal parameters depends on the usable amount of memory.
- If a macro call includes excess arguments, a warning will be output.
- If a macro is called before it has been defined, the compiler outputs an error.
- If a currently defined macro is called in a macro body, the CC-RL outputs the error message.
- If there are multiple formal parameters having the same name, an error will occur.
- Numeric constants or Symbol can be specified as an argument in a macro call; if another type of argument is specified, an error will be output.
- A line of a sentence can be designated in the macro-body. Such as operand can't designate the part of the sentence.

[Example]

```
ADMAC   .MACRO  PARA1, PARA2  ; Macro definition
        MOV     A, #PARA1
        ADD     A, #PARA2
        .ENDM

        ADMAC   0x10, 0x20    ; Macro call
```

.LOCAL

Defines the specified symbol name as a local symbol that is valid only within the macro body where it is defined.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.LOCAL	*symbol-name*[, ...]	[; *comment*]

[Function]

- Defines the specified symbol name as a local symbol that is valid only within the macro body where it is defined.

[Description]

- If a macro that defines a symbol within the macro body is referenced more than once, the assembler will output a double definition error for the symbol.
  By using the .LOCAL directive, you can reference (or call) a macro, which defines symbol(s) within the macro body, more than once.
- The maximum number of symbol names depends on the usable amount of memory.
- The .LOCAL directive can only be used within a macro definition or in "#pragma inline_asm" in a C source program.
- The .LOCAL directive should be used before the symbol specified in the operand field is referenced. Using this directive at the beginning of a macro body is recommended.
- All symbol names defined through the .LOCAL directive within a module should be different. A single name cannot be assigned to multiple local symbols used in each macro body.
- A symbol defined through the .LOCAL directive cannot be referenced from outside of the macro.
- Reserved words cannot be defined as symbols. When the same symbol as a user-defined symbol is specified, the definition through the .LOCAL directive takes priority.
- If a symbol having the same name as a formal parameter for a macro definition is defined as a local symbol in the definition of the macro through the .LOCAL directive, an error will occur.

[Example]

```
m1      .MACRO  par
        .LOCAL  AA, BB
AA:     .DB4    AA
BB:     .DB4    par
        .ENDM
m1      10
m1      20
```

The expansion is as follows.

```
.LL00000000:    .DB4    .LL00000000
.LL00000001:    .DB4    10
.LL00000002:    .DB4    .LL00000002
.LL00000003:    .DB4    20
```

.REPT

Tells the assembler to repeatedly expand a series of statements described between this directive and the .ENDM directive the number of times equivalent to the value of the absolute-expression specified in the operand field.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.REPT	*absolute-expression*	[; *comment*]
	:		
	.ENDM		[; *comment*]

[Function]

- The .REPT directive tells the assembler to repeatedly expand a series of statements described between this directive and the .ENDM directive (called the REPT-ENDM block) the number of times equivalent to the value of the absolute-expression specified in the operand field.

[Description]

- If no corresponding .ENDM directive is found for a .REPT directive in the same file, an error will occur
- If the result of evaluating the absolute-expression is negative, the CC-RL outputs the message.

[Example]

```
    .REPT    3
    INC      B
    DEC      C
    .ENDM
```

The code is expanded as shown below after assembling.

```
    INC      B
    DEC      C
    INC      B
    DEC      C
    INC      B
    DEC      C
```

.IRP

Tells the assembler to repeatedly expand a series of statements described between .IRP directive and the .ENDM directive the number of times equivalent to the number of actual parameters while replacing the formal parameter with the actual parameters (from the left, the order) specified in the operand field.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.IRP :	*formal-parameter*[*actual-parameter*[, ...]]	[; *comment*]

[Function]

- The .IRP directive tells the assembler to repeatedly expand a series of statements described between this directive and the .ENDM directive (called the IRP-ENDM block) the number of times equivalent to the number of actual parameters while replacing the formal parameter with the actual parameters (from the left, the order) specified in the operand field.

[Description]

- If no corresponding .ENDM directive is found for an .IRP directive within a single file, an error will occur.
- The maximum number of arguments depends on the usable amount of memory.

[Example]

```
    .IRP    PAR 0x10, 0x20, 0x30
    ADD     A, #PAR
    MOV     [DE], A
    .ENDM
```

The code is expanded as shown below after assembling.

```
    ADD     A, #0x10
    MOV     [DE], A
    ADD     A, #0x20
    MOV     [DE], A
    ADD     A, #0x30
    MOV     [DE], A
```

.EXITM

.EXITM directive skips the repetitive assembly of the .REPT and .IRP directives enclosing this directive at the innermost position.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.EXITM		[; *comment*]

[Function]

- .EXITM directive skips the repetitive assembly of the .REPT and .IRP directives enclosing this directive assembly of the innermost .REPT and .IPR directives enclosing this directive.

[Description]

- If .EXITM directive is not enclosed by .REPT and .IRP directives, the CC-RL outputs the message.
- The conditional assembly control instructions enclosing the .EXITM directive cannot be written between the .EXITM directive and .ENDM directive. ([V1.01 only])

[Example]

```
    .REPT   3
    .REPT   2
    INC     B
    .EXITM
    .ENDM
    DEC     C
    .ENDM
```

The code is expanded as shown below after assembling.

```
    INC     B
    DEC     C
    INC     B
    DEC     C
    INC     B
    DEC     C
```

.EXITMA

The .EXITMA directive skips the repetitive assembly of the outermost .REPT and .IRP directives enclosing this directive.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.EXITMA		[; *comment*]

[Function]

- The .EXITMA directive skips the repetitive assembly of the outermost .REPT and .IRP directives enclosing this directive.

[Description]

- If .EXITMA directive is not enclosed by .REPT and .IRP directives, the CC-RL outputs the message.
- The conditional assembly control instructions enclosing the .EXITMA directive cannot be written between the .EXITMA directive and .ENDM directive. ([V1.01 only])

[Example]

```
    .REPT   3
    .REPT   2
    INC     B
    .EXITMA
    .ENDM
    DEC     C
    .ENDM
```

The code is expanded as shown below after assembling.

```
    INC     B
```

.ENDM

Instructs the assembler to terminate the execution of a series of statements defined as the functions of the macro.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.ENDM		[; *comment*]

[Function]

- The .ENDM directive instructs the assembler to terminate the execution of a series of statements defined as the functions of the macro.

[Description]

- If the .MACRO, .REPT, or .IRP directive corresponding to .ENDM directive does not exist, the CC-RL outputs the message then stops assembling.

[Example]

.MACRO - .ENDM

```
ADMAC   .MACRO   PARA1, PARA2
        MOV      A, #PARA1
        ADD      A, #PARA2
        .ENDM
```

.REPT - .ENDM

```
        .REPT    3
        INC      B
        DEC      C
        .ENDM
```

.IRP - .ENDM

```
        .IRP     PAR 0x10, 0x20, 0x30
        ADD      A, #PAR
        MOV      [DE], A
        .ENDM
```

5.2.8 Branch directives

Branch directives are generated by the compiler and expanded into instructions by the assembler.
The following branch directives are available.

Table 5.24 Branch Directives

Directive	Overview
.Bcond	The assembler expands a .Bcond directive into instructions

.B*cond*

The assembler expands a .B*cond* directive into instructions

Remark .BT, .BF, .BC, .BNC, .BZ, .BNZ, .BH, and .BNH are collectively called .B*cond*.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.B*cond*	*label*	[; *comment*]

- In the above syntax, .BC, .BNC, .BZ, .BNZ, .BH, and .BNH can be specified as .B*cond*.
  A label can be specified in the form of !LABEL, $!LABEL, or !!LABEL.

Symbol field	Mnemonic field	Operand field	Comment field
	.B*cond*	*bit-term, label*	[; *comment*]

- In the above syntax, .BT, and .BF can be specified as .B*cond*.
  A label can be specified in the form of !LABEL, $!LABEL, or !!LABEL.

Symbol field	Mnemonic field	Operand field	Comment field
	.B*cond*	*label1, label2*	[; *comment*]

- In the above syntax, .BC, .BNC, .BZ, .BNZ, .BH, and .BNH can be specified as .B*cond*.
  For label1 and label2, the following combinations are allowed.

Label1	Label2
!LABEL1	$LABEL2
!LABEL1	!LABEL2
$!LABEL1	$LABEL2
$!LABEL1	$!LABEL2
$!LABEL1	!!LABEL2
!!LABEL1	$LABEL2
!!LABEL1	$!LABEL2
!!LABEL1	!!LABEL2

Symbol field	Mnemonic field	Operand field	Comment field
	.B*cond*	*bit-term, label1, label2*	[; *comment*]

- In the above syntax, .BT, and .BF can be specified as .B*cond*.
  For label1 and label2, the following combinations are allowed.

Label1	Label2
!LABEL1	!LABEL2
$!LABEL1	$!LABEL2
$!LABEL1	!!LABEL2

Label1	Label2
!!LABEL1	$!LABEL2
!!LABEL1	!!LABEL2

[Function]

- The assembler expands a .B*cond* directive into instructions.
- The .B*cond* directive is provided to be output by the compiler; the user must not use this directive.

[Description]

- .B*cond* directives are generated by the compiler, and the assembler expands them as follows.
  N*cond* indicates a condition opposite to *cond*.

Before Expansion	After Expansion
`.Bcond label`	`SKNcond` `BR label`

`.Bcond bit-term`	`BNcond bit-term, $temp`
	`BR label`
	`temp:`

`.Bcond label1, label2`	`SKNcond`
	`BR label1`
	`BR label2`

`.Bcond bit-term, label1, label2`	`Bcond bit-term, $temp`
	`BR label2`
	`temp:`
	`BR label1`

5.2.9 Machine-Language Instruction Set

The following instructions are not included in the RL78 instruction set but the compiler replaces them with available instructions. If any instruction that is included in neither the following list nor the instruction set, an error will occur.

Before Instruction Operand is Replaced	After Instruction Operand is Replaced
MOV [DE],#byte	MOV [DE+0],#byte
MOV [HL],#byte	MOV [HL+0],#byte
MOV ES:[DE],#byte	MOV ES:[DE+0],#byte
MOV ES:[HL],#byte	MOV ES:[HL+0],#byte
MOVS [HL],X	MOVS [HL+0],X
MOVS ES:[HL],X	MOVS ES:[HL+0],X
CMPS X,[HL]	CMPS X,[HL+0]
CMPS X,ES:[HL]	CMPS X,ES:[HL+0]
ADDW AX,[HL]	ADDW AX,[HL+0]
ADDW AX,ES:[HL]	ADDW AX,ES:[HL+0]
SUBW AX,[HL]	SUBW AX,[HL+0]
SUBW AX,ES:[HL]	SUBW AX,ES:[HL+0]
CMPW AX,[HL]	CMPW AX,[HL+0]
CMPW AX,ES:[HL]	CMPW AX,ES:[HL+0]
INC [HL]	INC [HL+0]
INC ES:[HL]	INC ES:[HL+0]
DEC [HL]	DEC [HL+0]
DEC ES:[HL]	DEC ES:[HL+0]
INCW [HL]	INCW [HL+0]
INCW ES:[HL]	INCW ES:[HL+0]
DECW [HL]	DECW [HL+0]
DECW ES:[HL]	DECW ES:[HL+0]

5.3 Control Instructions

Control instructions provide instructions for assembler operation.

5.3.1 Outline

Control instructions provide detailed instructions for assembler operation and so are written in the source.
Control instruction categories are displayed below.

Table 5.25 Control Instruction List

Control Instruction Type	Control Instruction
File input control instructions	INCLUDE, BINCLUDE
Mirror source area reference control instructions	MIRROR
Assembler control instructions	NOWARNING, WARNING
Conditional assembly control instructions	IFDEF, IFNDEF, IF, IFN, ELSEIF, ELSEIFN, ELSE, ENDIF

As with directives, control instructions are specified in the source.

5.3.2 File input control instructions

Using the file input control instruction, the CC-RL can input an assembly source file or binary file to a specified position. The following file input control instructions are available.

Table 5.26 File Input Control Instructions

Control Instruction	Overview
INCLUDE	Inputs an assembly language source file
BINCLUDE	Inputs a binary file

INCLUDE

Inputs an assembly language source file.

[Syntax]

```
[Δ]$[Δ]INCLUDE[Δ](file-name)[Δ][;comment]]
[Δ]$[Δ]INCLUDE[Δ]"file-name"[Δ][;comment]
```

[Function]

- The INCLUDE control instruction tells the assembler to insert and expand the contents of a specified file beginning on a specified line in the source program for assembly.

[Description]

- The search pass of an INCLUDE file can be specified with the option (-I).
- The assembler searches INCLUDE file read paths in the following sequence:
  (a) Folder specified by the option (-I)
  (b) Folder in which the source file exists
  (c) Currently folder
- The INCLUDE file can do nesting (the term "nesting" here refers to the specification of one or more other INCLUDE files in an INCLUDE file).
- The maximum nesting level for include files is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.
- If the specified INCLUDE file cannot be opened, the CC-RL outputs the message and stops assembling.
- If an include file contains a block from start to finish, such as a section definition directive, macro definition directive, or conditional assembly control instruction, then it must be closed with the corresponding code. If there is no corresponding code or it is not closed, then an error will occur.

BINCLUDE

Inputs a binary file.

[Syntax]

```
[Δ]$[Δ]BINCLUDE[Δ](file-name)[Δ][;comment]]
[Δ]$[Δ]BINCLUDE[Δ]"file-name"[Δ][;comment]
```

[Function]

- Assumes the contents of the binary file specified by the operand to be the result of assembling the source file at the position of this control instruction.

[Description]

- This control instruction handles the entire contents of the binary files.
- The specified binary file is searched for in the same order as for the $INCLUDE control instruction.
- If the specified binary file cannot be opened, the CC-RL outputs the message and stops assembling.

5.3.3 Mirror source area reference control instructions

Using the mirror source area reference control instruction enables reference to external reference names in the sections allocated at mirror source area addresses.

The following mirror source area reference control instructions are available.

Table 5.27 Mirror Source Area Reference Control Instructions

Control Instruction	Overview
MIRROR	Declares that an external reference name is allocated to the mirror source area

MIRROR

Declares that an external reference name is allocated to the mirror source area.

[Syntax]

```
[Δ]$[Δ]MIRROR[Δ]external reference-name[Δ][;comment]
```

[Function]

- Assumes that the specified external reference name is allocated to the mirror source area and generates an instruction to refer to the data.

[Description]

- This control instruction is used to reference the external reference name that is allocated to the mirror source area.

5.3.4 Assembler control instructions

The assembler control instruction can be used to control the processing performed by the assembler.
The following assembler control instructions are available.

Table 5.28 Assembler Control Instructions

Control Instruction	Overview
NOWARNING	Does not output warning messages
WARNING	Output warning messages

NOWARNING

Does not output warning messages.

[Syntax]

```
[Δ]$[Δ]NOWARNING[Δ][;comment]
```

[Function]

- This control instruction suppresses outputting warning messages for this control instruction and the subsequent instructions.

WARNING

Output warning messages.

[Syntax]

```
[Δ]$[Δ]WARNING[Δ][;comment]
```

[Function]

- This control instruction outputs warning messages for this control instruction and the subsequent instructions (cancels the NOWARNING specification).

5.3.5 Conditional assembly control instructions

Using conditional assembly control instruction, the CC-RL can control the range of assembly according to the result of evaluating a conditional expression.

The following conditional assembly control instructions are available.

Table 5.29 Conditional Assembly Control Instructions

Control Instruction	Overview
IFDEF	Control based on symbol (assembly performed when the symbol is defined)
IFNDEF	Control based on symbol (assembly performed when the symbol is not defined)
IF	Control based on absolute expression (assembly performed when the value is true)
IFN	Control based on absolute expression (assembly performed when the value is false)
ELSEIF	Control based on absolute expression (assembly performed when the value is true)
ELSEIFN	Control based on absolute expression (assembly performed when the value is false)
ELSE	Control based on absolute expression/symbol
ENDIF	End of control range

The maximum number of nest level of the conditional assembly control instruction is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.

IFDEF

Control based on symbol (assembly performed when the symbol is defined).

[Syntax]

```
[Δ]$[Δ]IFDEFΔswitch-name[Δ][;comment]
```

[Function]

- If the symbol specified by the switch name is defined
  - (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.
  - (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.
- If the symbol specified by the switch name is not defined
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

[Description]

- The rules of describing switch names are the same as the conventions of symbol description "(3) Symbol".
- The maximum number of nest level of the conditional assembly control instruction is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.

IFNDEF

Control based on symbol (assembly performed when the symbol is not defined).

[Syntax]

```
[Δ]$[Δ]IFNDEFΔswitch-name[Δ][;comment]
```

[Function]

- If the symbol specified by the switch name is defined
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

- If the symbol specified by the switch name is not defined

  (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.

  (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.

[Description]

- The rules of describing switch names are the same as the conventions of symbol description "(3) Symbol".
- The maximum number of nest level of the conditional assembly control instruction is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.

IF

Control based on absolute expression (assembly performed when the value is true).

[Syntax]

```
[Δ]$[Δ]IFΔabsolute-expression[Δ][;comment]
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠ 0).
  (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.
  (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.
- If the absolute expression is evaluated as being false (= 0).
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

[Description]

- The maximum number of nest level of the conditional assembly control instruction is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.

IFN

Control based on absolute expression (assembly performed when the value is false).

[Syntax]

```
[Δ]$[Δ]IFNΔabsolute-expression[Δ][;comment]
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠ 0).
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.
- If the absolute expression is evaluated as being false (= 0).
  (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.
  (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.

[Description]

- The maximum number of nest level of the conditional assembly control instruction is 4,294,967,294 (=0xFFFFFFFE) (theoretical value). The actual number that can be used depends on the amount of memory, however.

ELSEIF

Control based on absolute expression (assembly performed when the value is true).

[Syntax]

```
[Δ]$[Δ]ELSEIFΔabsolute-expression[Δ][;comment]
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠ 0).
  (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.
  (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.
- If the absolute expression is evaluated as being false (= 0).
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

ELSEIFN

Control based on absolute expression (assembly performed when the value is false).

[Syntax]

```
[Δ]$[Δ]ELSEIFNΔabsolute-expression[Δ][;comment]
```

[Function]

- If the absolute expression specified by the operand is evaluated as being true (≠ 0).
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.
- If the absolute expression is evaluated as being false (= 0).
  (a) If this control instruction and the corresponding ELSEIF, ELSEIFN, or ELSE control instruction exist, assembles the block enclosed within this control instruction and the corresponding control instruction.
  (b) If none of the corresponding control instruction detailed above exist, assembles the block enclosed within this control instruction and the corresponding ENDIF control instruction.

ELSE

Control based on absolute expression/symbol.

[Syntax]

```
[Δ]$[Δ]ELSE[Δ][;comment]
```

[Function]

- If the specified switch name is not defined by the IFDEF control instruction, if the absolute expression of the IF, or ELSEIF control instruction is evaluated as being false (= 0), or if the absolute expression of the IFN, or ELSEIFN control instruction is evaluated as being true (≠ 0), assembles the arrangement of statements (block) enclosed within this control instruction and the corresponding ENDIF control instruction.

ENDIF

End of control range.

[Syntax]

```
[Δ]$[Δ]ENDIF[Δ][;comment]
```

[Function]

Indicates the end of the control range of a conditional assembly control instruction.

5.4 Macro

This section explains how to use macros.
This is very convenient function to describe serial instruction group for number of times in the program.

5.4.1 Outline

This macro function is very convenient function to describe serial instruction group for number of times in the program.
Macro function is the function that is deployed at the location where serial instruction group defined as macro body is referred by macros as per .MACRO, .ENDM directives.
Macro differs from subroutine as it is used to improve description of the source.
Macro and subroutine has features respectively as follows. Use them effectively according to the respective purposes.

- Subroutine
  Process required many times in program is described as one subroutine. Subroutine is converted in machine language only once by assembler.
  Subroutine/call instruction (generally instruction for argument setting is required before and after it) is described only in subroutine reference. Consequently, memory of program can be used effectively by using subroutine.
  It is possible to draw structure of program by executing subroutine for process collected serially in program (Because program is structured, entire program structure can be easily understood as well setting of the program also becomes easy.).

- Macro
  Basic function of macro is to replace instruction group.
  Serial instruction group defined as macro body by .MACRO, .ENDM directives are deployed in that location at the time of referring macro. Assembler deploys macro/body that detects macro reference and converts the instruction group to machine language while replacing temporary parameter of macro/body to actual parameter at the time of reference.
  Macro can describe a parameter.
  For example, when process sequence is the same but data described in operand is different, macro is defined by assigning temporary parameter in that data. When referring the macro, by describing macro name and actual parameter, handling of various instruction groups whose dercription is different in some parts only is possible.

Subroutine technique is used to improve efficiency of coding for macro to use to draw structure of program and reducing memory size.

5.4.2 Usage of macro

A macro is described by registering a pattern with a set sequence and by using this pattern. A macro is defined by the user. A macro is defined as follows. The macro body is enclosed by ".MACRO" and ".ENDM".

```
ADDINT8 .MACRO  PARA1, PARA2  ;The following two statements constitute the macro body.
        MOV     A, #PARA1
        ADD     A, #PARA2
        .ENDM
```

If the following description is made after the above definition has been made, the macro is replaced by a code that "adds 0x10 and 0x20".

```
ADDINT8 0x10, 0x20
```

In other words, the macro is expanded into the following codes.

```
MOV     A, #0x10
ADD     A, #0x20
```

5.4.3 Nesting macro definitions

A .MACRO directive cannot be written between a .MACRO, .REPT, or .IRP directive and its corresponding .ENDM directive.

5.4.4 Nesting macro references

The maximum nesting level of macro references is theoretically 4,294,967,294 (= 0xFFFFFFFE), but the memory size limits the actual maximum level.

5.4.5 Macro operator

This section describes the concatenation symbols "?", which are used to link strings in macros.

- The concatenation "?" concatenates one character or one character string to another within a macro body.
  At macro expansion, the character or character string on the left of the concatenation is concatenated to the character or character string on the right of the sign. The "?" itself disappears after concatenating the strings.
- The symbols before and after the concatenation symbol "?" in the symbols of a macro definition can be recognized as formal parameters or local symbols, and concatenation symbols can also be used as delimiter symbols. At macro expansion, formal parameters or local symbols before and after "?" in the symbols are evaluated before they are concatenated in the symbols.
- Cautions
  - The character "?" can only be used as a concatenation symbol in a macro definition.
  - The "?" in a character string and comment is simply handled as data.

5.4.6 Error processing

Errors regarding the correspondence between the .MACRO, .REPT, or .IRP directive and the .ENDM directive are output in the following cases.

- If no corresponding .ENDM directive is found for a .MACRO, .REPT, or .IRP directive until the end of the source program, an error will be output at the end of the source program.
- If no corresponding .MACRO, .REPT, or .IRP directive is found for an .ENDM directive, an error will be output.
- If an error is found in the line where a .MACRO directive is defined, the line will be ignored and processing will continue. Therefore, the macro name will not be defined and if the macro name is referenced, an error will be output.

5.5 Using SFR Symbols and Extended SFR Symbols

When a device file is read, SFR symbols and extended SFR symbols can be used in the assembly source code.
For the SFR symbols and extended SFR symbols that can be used in the assembly source code, see the user's manual of the device.
SFR symbols and extended SFR symbols are handled in the same way as constants (addresses of the specified SFRs), but note the following.

- Uppercase and lowercase letters are not distinguished for SFR symbols and extended SFR symbols.
- SFR symbols and extended SFR symbols, which are abbreviations for SFR and extended SFR names and not actually symbols, cannot be defined as symbols in program code.
- "!" needs to be at the beginning of an extended SFR symbol in an instruction operand.
  "!" can be omitted when the assembler transition support option (-convert_asm) is specified.

See also the descriptions of sfr, sfrp, and addr16 in "5.1.16 Operand characteristics".

5.6 Reserved Words

The assembler has reserved words. Reserved word cannot be used in name, label, section name, macro name. If a reserved word is specified, the CC-RL outputs the message. Reserved word doesn't distinguish between uppercase and lowercase.

The reserved words are as follows.

- Instructions (such as add, sub, and mov)
- Directives
- Control instructions
- Register names, Internal register name

5.7 Assembler Generated Symbols

The following is a list of symbols generated by the assembler for use in internal processing.
Symbols with the same names as the symbols below cannot be used.
The assembler does not output object files for symbols starting with a period ("."), treating these as symbols for internal processing.

Table 5.30 Assembler Generated Symbols

Symbol Name	Explanation
.LL00000000 to .LLFFFFFFFF	.LOCAL directive generation local symbols
.text_AT*start-address*	.CSEG directive generation section name
.data_AT*start-address*	.DSEG directive generation section name
.bss_AT*start-address*	
.bit_AT*start-address*	.BSEG directive generation section name
*section-name*_AT*start-address*	.ORG directive generation section name
.BR_TEMP@*n*	Label for branch instruction expansion
.LM*n*_*n* (*n* : 0 to 4294967294)	Assembler debugging information symbol (start or end of section or function) - Example : .LM0_1
.G*n* (*n* : 0 to 4294967294)	Assembler debugging information symbol (start or end of .debug_info, line, frame, or loc)
.Gar*n* (*n* : 0 to 4294967294)	Assembler debugging information symbol (start or end of .debug_pubnames section)
.Gpu*n* (*n* : 0 to 4294967294)	Assembler debugging information symbol (start or end of .debug_aranges section)
@$IMM_*constant value*	Local symbol indicating a constant value [V1.02 or later]

6. SECTION SPECIFICATIONS

In an embedded application such as allocating program code from certain address or allocating by division, it is necessary to pay attention in the memory allocation.

To implement the memory allocation as expected, program code or data allocation information should be specified in optimizing linker.

Sections which are units for memory allocation are described in this chapter.

6.1 Sections

A section is the basic unit making up programs (area to which programs or data are allocated). For example, program code is allocated to a text-attribute section and variables that have initial values are allocated to a data-attribute section. In other words, different types of information are allocated to different sections.

Section names can be specified within source file. In C language, they can be specified using a #pragma section directive and in assembly language they can be specified using section definition directives.

Even if the #pragma directive is not used to specify a section, however, allocation by the compiler to a particular section may already be set as the default setting in the program code or data (variables).

6.1.1 Section name

The following table lists the names, relocation attribute of these reserved sections.

Table 6.1 Reserved Sections

Default Section Name	Relocation Attribute	Description
.callt0	CALLT0	Section for the table to call the callt function
.text	TEXT	Section for code (allocated to the near area)
.textf	TEXTF	Section for code (allocated to the far area)
.textf_unit64kp	TEXTF_UNIT64KP	Section for code (section is allocated so that the start address is an even address and the section does not exceed the (64 Kbytes - 1) boundary)
.const	CONST	ROM data (allocated to the near area) (within the mirror area)
.constf	CONSTF	ROM data (allocated to the far area)
.data	DATA	Section for near initialized data (with initial value)
.dataf	DATAF	Section for far initialized data (with initial value)
.sdata	SDATA	Section for initialized data (with initial value, variable allocated to saddr)
.bss	BSS	Section for data area (without initial value, allocated to the near area)
.bssf	BSSF	Section for data area (without initial value, allocated to the far area)
.sbss	SBSS	Section for data area (without initial value, variable allocated to saddr)
.option_byte	OPT_BYTE	Dedicated section for user option byte and on-chip debugging specification
.security_id	SECUR_ID	Dedicated section for security ID specification
.flash_security_id	FLASH_SECUR_ID	Section specific for flash programmer security ID specification

Default Section Name	Relocation Attribute	Description
.vect<vector table address>	AT	Interrupt vector table If the -split_vect option is specified, a section is generated based on ".vect<vector table address>". The vector table address is in hexadecimal notation.

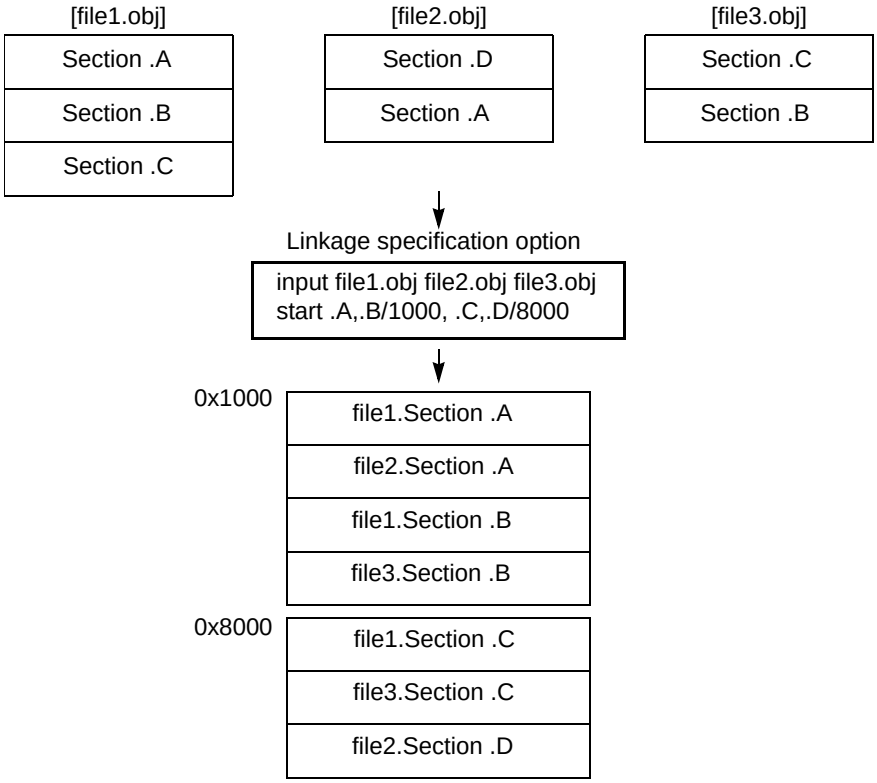
6.1.2 Section concatenation

The optimizing linker (hereafter abbreviated "rlink") concatenates identical sections in the input relocatable files, and allocates them to the address specified by the -start option.

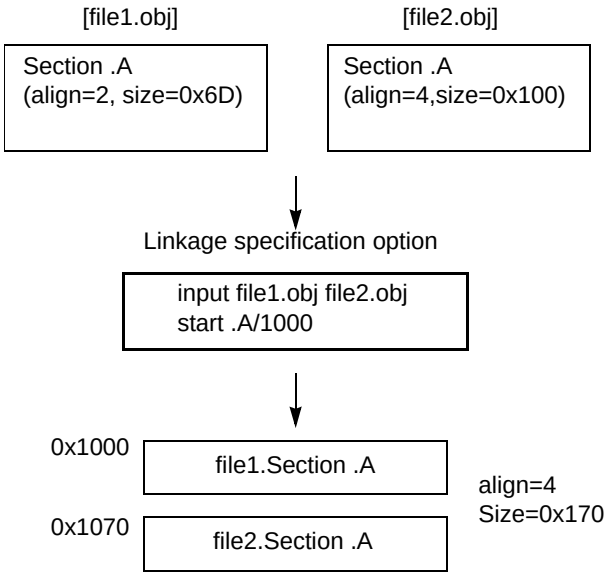
Remark See "-STARt" for details of -start option.

(1) Section allocation via the -start option

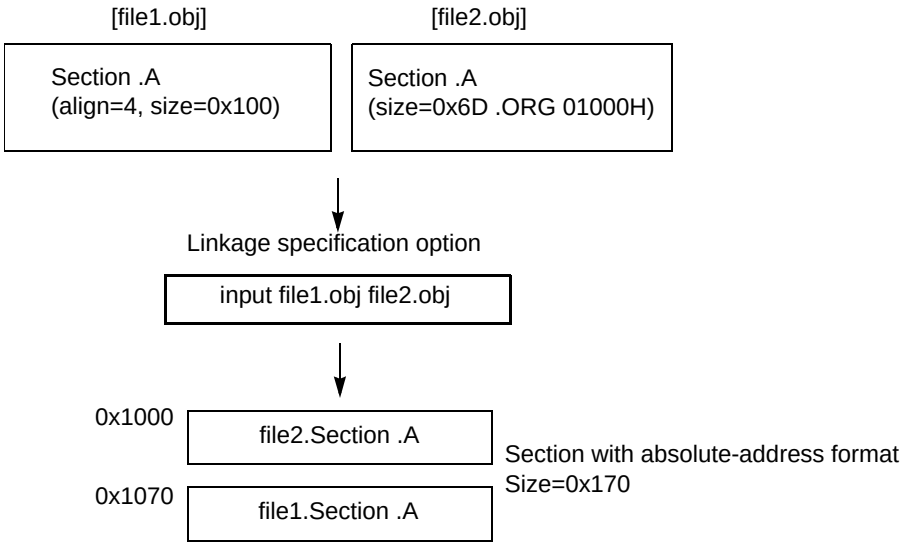
(a) Sections in different files with the same name are concatenated and allocated in the order of file input.



(b) Sections with the same name but different alignments are concatenated after alignment adjustment. The alignment is adjusted to that of the section with the largest alignment.

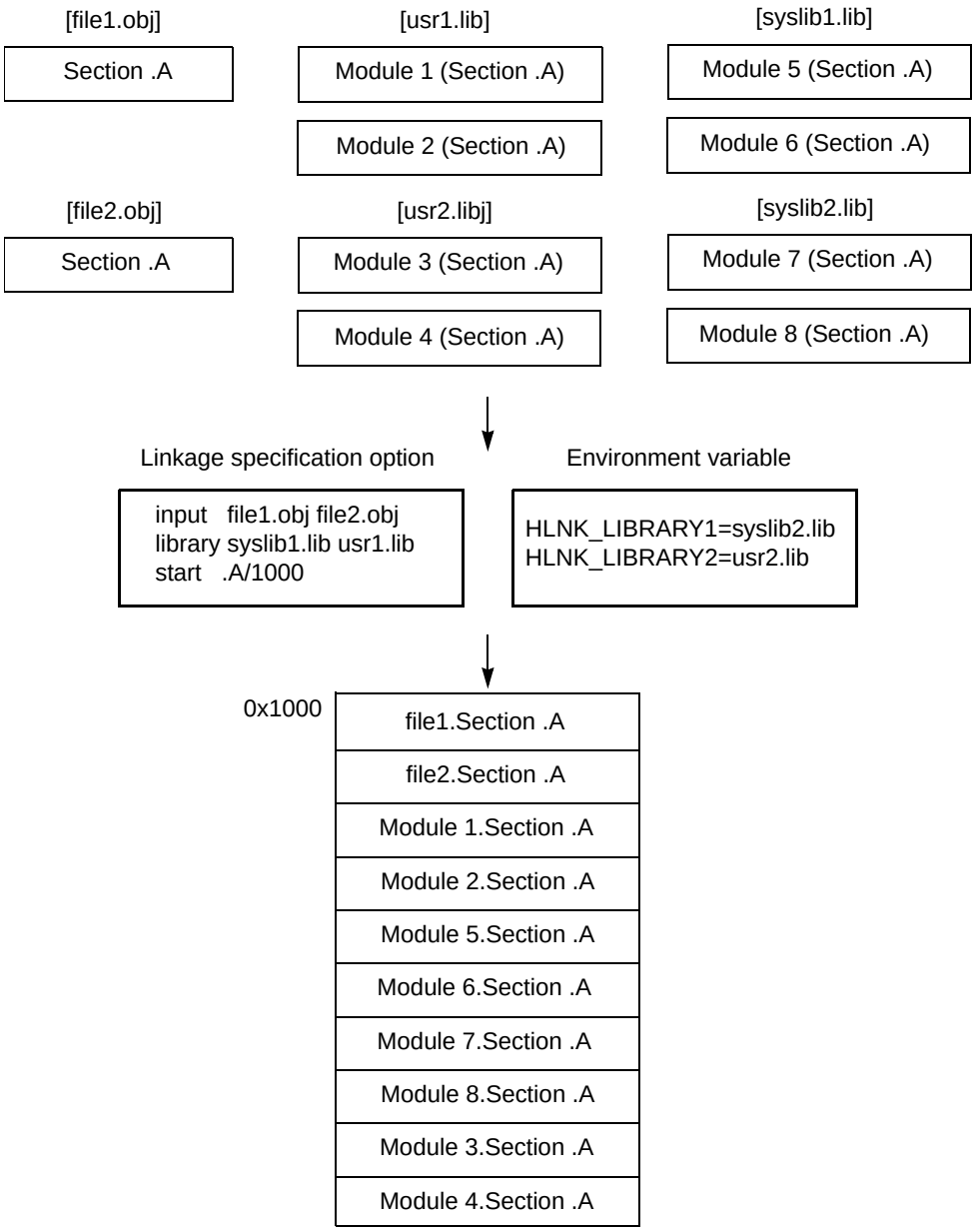


(c) If sections with the same name include both absolute-address format and relative-address format, then the sections with relative-address format are concatenated after the sections with absolute-address format.



(d) The rules for ordering of concatenation for sections with the same name are indicated below, highest priority to lowest.

- Order in which input files are specified via the input option or on the command line
- Order in which user libraries are specified via the library option and order of modules input in the library
- Order in which system libraries are specified via the library option and order of modules input in the library
- Order in which environment variable (HLNK_LIBRARY1 to 3) libraries are specified and order of modules input in the library


[file1.obj]
Section .A
[usr1.lib]
Module 1 (Section .A)
Module 2 (Section .A)
[syslib1.lib]
Module 5 (Section .A)
Module 6 (Section .A)
[file2.obj]
Section .A
[usr2.libj]
Module 3 (Section .A)
Module 4 (Section .A)
[syslib2.lib]
Module 7 (Section .A)
Module 8 (Section .A)
Linkage specification option
input file1.obj file2.obj
library syslib1.lib usr1.lib
start .A/1000
Environment variable
HLNK_LIBRARY1=syslib2.lib
HLNK_LIBRARY2=usr2.lib
0x1000
file1.Section .A
file2.Section .A
Module 1.Section .A
Module 2.Section .A
Module 5.Section .A
Module 6.Section .A
Module 7.Section .A
Module 8.Section .A
Module 3.Section .A
Module 4.Section .A

6.2 Special Symbol

The optimizing linker generates symbols indicating the start and end of each output section during linkage or generates symbols by specifying options.

6.2.1 Symbols generated regardless of option specifications

The optimizing linker generates symbols indicating the start and end of a section, such as, .data and .sdata.

Start symbol: __s*section-name*
End symbol: __e*section-name*

For example, the symbol names of the start symbol and end symbol of the .data section become __s.data and __e.data, respectively. Note that a symbol will not be generated when the defined symbol name is the same with a symbol generated by the optimizing linker.

6.2.2 Symbols generated by option specifications

The optimizing linker generates the following symbols using option specifications.

Table 6.2 Symbols generated by option specifications

Option	Symbol Name	Explanation
-hide	$CNCL_*n* n : 1 - 4294967295 Example) $CNCL_1	The local symbol name in the output file is changed to $CNCL_n.
-user_opt_byte -ocdbg	.option_byte	The section name ".option_byte" is generated when the device file is specified with the section name ".option_byte" not existing in the source file. Note that the user option byte value and control value for on-chip debugging operation are set in the above section.
-security_id	.security_id	The section name ".security_id" is generated when the device file is specified with the section name ".security_id" not existing in the source file. Note that the security ID value are set in the above section.
-device	__STACK_ADDR_START __STACK_ADDR_END	A consecutive area not used in the RAM area acquired from the device file is searched for and the following values are set to __STACK_ADDR_START and __STACK_ADDR_END. __STACK_ADDR_START: Maximum address of the area + 1 __STACK_ADDR_END: Minimum address of the area However, if the -SELF, -SELFW, -OCDTR, -OCDTRW, -OCDHPI, or -OCDHPIW option is specified, the area is set from the RAM area excluding the saddr area.
	__RAM_ADDR_START __RAM_ADDR_END	The RAM area is acquired from the device file, and the following values are set to __RAM_ADDR_START and __RAM_ADDR_END. __RAM_ADDR_START: Start address of RAM area __RAM_ADDR_END: End address of RAM area + 1
-debug_monitor	.monitor1 .monitor2	When the device file is specified while section names ".monitor1" and ".monitor2" do not exist in the source file, section names ".monitor1" and ".monitor2" are generated.
-rrm	.rrm	[V1.13 or later] If section name ".rrm" does not exist in the source file, section name ".rrm" is generated.

7. LIBRARY FUNCTION SPECIFICATIONS

This chapter describes the library functions provided in the CC-RL.

7.1 Supplied Libraries

The CC-RL provides the following libraries.

Table 7.1 Supplied Libraries

Supplied Libraries	Library Name	Outline
Standard library [V1.03 or later] (except for calloc, free, malloc, and realloc)	rl78nm4s.lib rl78nm4s99.lib [V1.07 or later]	For an 8-bit or 16-bit CPU without extended instructions and arithmetic units
	rl78cm4s.lib rl78cm4s99.lib [V1.07 or later]	For a 16-bit CPU using division/multiplication and multiply-accumulate units
	rl78em4s.lib rl78em8s.lib rl78em4s99.lib [V1.07 or later] rl78em8s99.lib [V1.07 or later]	For a 16-bit CPU using division/multiplication extended instructions
[V1.03 or later] Standard library (calloc, free, malloc, realloc)	malloc_n.lib	malloc library for normal usage
	malloc_s.lib	malloc library for security facility [Professional Edition only]
Runtime library	rl78nm4r.lib	For an 8-bit or 16-bit CPU without extended instructions and arithmetic units
	rl78cm4r.lib	For a 16-bit CPU using division/multiplication and multiply-accumulate units
	rl78em4r.lib rl78em8r.lib	For a 16-bit CPU using division/multiplication extended instructions
Startup routine	cstart.asm	Startup routine

In order to pass the far pointer to a standard library function that has a variable pointer as a parameter, the user has to call the function for the far variable pointer whose function name starts with "_COM_". Note that when the -far_rom option is specified, the function macros of the header file are valid and the function for the far variable pointer is automatically called.

7.2 Rule for Naming Libraries

When the standard library is used in an application, include the related header files to use the library function.

The runtime library function is a routine that is automatically called by the CC-RL when a floating-point operation or integer operation is performed.

Refer these libraries using the optimizing linker option (-library). The type of library files used in a single project must be unified.

The rule for naming libraries is given below.

[V1.03 or later]

Since malloc library functions used for normal usage have different facilities from those used for the security facility, they become different libraries from the other standard libraries. The malloc libraries are common for RL78-S1, RL78-S2, and RL78-S3.

rl78<*muldiv*><*memory_model*><*float*><*standard*/*runtime*><*lang*>.lib
malloc_<*secure*>.lib (malloc library) [V1.03 or later]

<*muldiv*>
n : Without extended instructions and arithmetic units (for RL78-S1 core/RL78-S2 core)
c : Using division/multiplication and multiply-accumulate units (for RL78-S2 core)
e : Using division/multiplication extended instructions (for RL78-S3 core)
<*memory_model*>
m : Small model or medium model
<*float*>
4 : Single-precision floating-point number
8 : Double-precision floating-point number (supported only in devices with division/multiplication extended instructions)
<*standard/runtime*>
s : Standard library
r : Runtime library
<*secure*>
n: For normal usage
s: For security facility [Professional Edition only]
<*lang*>
None: For C90
99: For C99 [V1.07 or later]

7.3 Allocation Area of Libraries and Startup Routine

The sections for code of the library functions can be allocated to all areas. The startup routine can be allocated to only addresses 0x00000 to 0x0FFFF.

Section Name	Relocation Attribute	Description
.RLIB	TEXTF	Section for code of runtime libraries
.SLIB	TEXTF	Section for code of standard libraries
.data .bss .constf	DATA BSS CONSTF	Section for data of standard libraries
.text	TEXT	Startup routine

7.4 Header Files

The list of header files required for using the libraries of the CC-RL are listed below.
The macro definitions and function/variable declarations are described in each file.

Table 7.2 Header Files

File Name	Outline
assert.h	Header file for program diagnostics
ctype.h	Header file for character conversion and classification
errno.h	Header file for reporting error condition
float.h	Header file for floating-point representation and floating-point operation
inttypes.h (C99 only) [V1.07 or later]	Header file for format conversion of integer types
iso646.h (C99 only) [V1.06 or later]	Header file for alternative spellings of macro names
limits.h	Header file for quantitative limiting of integers
math.h	Header file for mathematical calculation
setjmp.h	Header file for non-local jump
stdarg.h	Header file for supporting functions having variable arguments
stdbool.h (C99 only) [V1.06 or later]	Header file for logical types and values
stddef.h	Header file for common definitions
stdint.h	Header file for integer type of the specified width
stdio.h	Header file for standard I/O
stdlib.h	Header file for general utilities
string.h	Header file for memory manipulation and character string manipulation
_h_c_lib.h	Header file for initialization

7.5 Library Function

This section explains Library Function.
When specifications differ between C90 and C99, (C90) is written for C90 library functions and (C99) is written for C99 library functions. [V1.07 or later]

When using a standard or mathematical library, the header file must be included.
When the -far_rom option is specified, in order to match the interface of the near/far attribute of the pointer, macro replacement is performed for the functions in the header file and then the function for the far variable pointer is called.
When one of the parameters is a pointer, the far pointer is used fixedly for format specification of the standard I/O functions and token division of the string operation functions. All other cases are dependent on the memory model and -far_rom option.
If the far pointer is passed as a parameter when small model or medium model is specified, the function for the far variable pointer is called.
In order to use a mathematical function of the float type when double-precision is specified, call a mathematical function for the float type, in which "f" is appended to the end of the function name.

7.5.1 Program diagnostic functions

Program diagnostic functions are as follows

Table 7.3 Program Diagnostic Function

Function/Macro Name	Outline
assert	Adds diagnostic features to the program

assert

Adds diagnostic features to the program.

[Classification]

Standard library

[Syntax]

```
#include <asssert.h>
void    assert(int expression);
```

[Description]

Adds diagnostic features to the program.

If expression is true, ends processing without returning a value. If expression is false, it outputs diagnostic information to the standard error file in the format defined by the compiler, and then calls the abort function.

The diagnostic information includes the program text of the parameters, the name of the source file, and the line number of the source.

When -lang=c99 is specified, the function name is also included in the diagnostic information. [V1.06 or later]

If you wish to disable the assert macro, include a #define NDEBUG statement before assert.h is loaded.

7.5.2 Character operation functions

Character operation functions are as follows.

Table 7.4 Character operation Functions

Function Name	Outline
isalnum	Identification of ASCII letter or numeral
isalpha	Identification of ASCII letter
isascii	Identification of ASCII code
isblank [V1.07 or later]	Identification of space or tab (C99)
iscntrl	Identification of control character
isdigit	Identification of decimal number
isgraph	Identification of display character other than space
islower	Identification of lower-case character
isprint	Identification of display character
ispunct	Identification of delimiter character
isspace	Identification of space/tab/carriage return/line feed/vertical tab/page feed
isupper	Identification of upper-case character
isxdigit	Identification of hexadecimal number
toascii	Judges if a character is an ASCII code
tolower	Conversion from upper-case to lower-case (not converted if argument is not in upper-case)
toupper	Conversion from lower-case to upper-case (not converted if argument is not in lower-case)

isalnum

Identification of ASCII letter or numeral

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isalnum(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isalpha

Identification of ASCII letter

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isalpha(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isascii

Identification of ASCII code

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isascii(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isblank [V1.07 or later]

Identification of space or tab

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far isblank(int c); (C99)
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

iscntrl

Identification of control character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       iscntrl(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isdigit

Identification of decimal number

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isdigit(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isgraph

Identification of display character other than space

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isgraph(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

islower

Identification of lower-case character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       islower(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isprint

Identification of display character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isprint(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

ispunct

Identification of delimiter character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       ispunct(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isspace

Identification of space/tab/carriage return/line feed/vertical tab/page feed

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isspace(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isupper

Identification of upper-case character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isupper(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

isxdigit

Identification of hexadecimal number

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       isxdigit(int c);
```

[Return value]

This function returns a value other than 0 if the value of argument *c* matches the respective description (i.e., if the result is true). If the result is false, 0 is returned.

toascii

Conversion to ASCII code

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       toascii(int c);
```

[Return value]

Returns the value of *c* with its lower seven bits masked.

[Description]

This function converts the value of *c* into the ASCII code. Bits (bits 7 to 15) other than the ASCII code range (bits 0 to 6) are set to 0.

tolower

Conversion from upper-case to lower-case

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       tolower(int c);
```

[Return value]

If isupper is true with respect to c, returns a character that makes islower true in response; otherwise, returns *c*.

[Description]

This function is a macro that converts uppercase characters into the corresponding lowercase characters and leaves the other characters unchanged.

toupper

Conversion from lower-case to upper-case

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int __far       toupper(int c);
```

[Return value]

If islower is true with respect to *c*, returns a character that makes isupper true in response; otherwise, returns *c*.

[Description]

This function is a macro that converts lowercase characters into the corresponding uppercase characters and leaves the other characters unchanged.

7.5.3 Functions for greatest-width integer types

The functions for greatest-width integer types are as follows.

Table 7.5 Functions for Greatest-Width Integer Types

Function Name	Outline
imaxabs [V1.07 or later]	Output of absolute value (intmax_t type) (C99)
imaxdiv [V1.07 or later]	Division (intmax_t type) (C99)
strtoimax [V1.07 or later]	Conversion of character string to integer (intmax_t type) and storing pointer to last character string (C99)
strtoumax [V1.07 or later]	Conversion of character string to integer (uintmax_t type) and storing pointer to last character string (C99)

imaxabs [V1.07 or later]

Obtains the absolute value (intmax_t type).

[Classification]

Standard library

[Syntax]

```
#include <inttypes.h>
intmax_t __far imaxabs(intmax_t j); (C99)
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |. If the input value of imaxabs is the smallest negative value, the same value is returned.

[Description]

This function obtains the absolute value of *j* (size of *j*), | *j* |. If *j* is a negative number, the result is the reversal of *j*. If *j* is not negative, the result is *j*.

imaxdiv [V1.07 or later]

Performs division of intmax_t type to obtain the quotient and remainder

[Classification]

Standard library

[Syntax]

#include <inttypes.h>
imaxdiv_t __far imaxdiv(intmax_t *numer*, intmax_t *denom*); (C99)

[Return value]

The structure holding the result of the division is returned. When divided by 0, -1 is set as quotient quot and *numer* is set as remainder rem.

[Description]

This function is used to divide a value of intmax_t type.
This function calculates the quotient (quot) and remainder (rem) resulting from dividing numerator *numer* by denominator *denom*, and stores these two integers as the members of the following structure imaxdiv_t.

```
typedef struct {
    intmax_t quot;
    intmax_t rem;
} imaxdiv_t;
```

When the value cannot be divided, the quotient of the result becomes an integer that is closest to the algebraical quotient and has a smaller absolute value than it.

strtoimax [V1.07 or later]

Converts a character string to an integer (intmax_t type) and stores the pointer to the last character string.

[Classification]

Standard library

[Syntax]

```
#include <inttypes.h>
intmax_t __far strtoimax(const char __near * restrict nptr, char __near * __near * restrict endptr, int base); (C99)
intmax_t __far _COM_strtoimax_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base); (C99)
```

[Return value]

If the partial character string has been converted, the converted value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, this function returns INTMAX_MAX or INTMAX_MIN and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into an intmax_t-type representation. If *base* is 0, the string is interpreted as the C radix representation. If *base* is in the range from 2 to 36, the string is interpreted as a radix. When *endptr* is not a null pointer, the pointer to the remaining strings that were not converted is set in *endptr*.

strtoumax [V1.07 or later]

Converts a character string to an integer (uintmax_t type) and stores the pointer to the last character string.

[Classification]

Standard library

[Syntax]

```
#include <inttypes.h>
uintmax_t __far strtoumax(const char __near * restrict nptr, char __near * __near * restrict endptr, int base); (C99)
uintmax_t __far _COM_strtoumax_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base); (C99)
```

[Return value]

If the partial character string has been converted, the converted value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, this function returns UINTMAX_MAX and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a uintmax_t-type representation. If *base* is 0, the string is interpreted as the C radix representation. If *base* is in the range from 2 to 36, the string is interpreted as a radix. When *endptr* is not a null pointer, the pointer to the remaining strings that were not converted is set in *endptr*.

7.5.4 Mathematical functions

Mathematical functions are as follows.

Table 7.6 Mathematical Functions

Function/Macro Name	Outline
fpclassify [V1.08 or later]	Classifies a floating-point number as NaN, infinite, normal, subnormal or zero (C99)
isfinite [V1.08 or later]	Determines whether a floating-point number has a finite value (zero, subnormal, or normal) (C99)
isinf [V1.08 or later]	Determines whether a floating-point number is an infinity (C99)
isnan [V1.08 or later]	Determines whether a floating-point number is a NaN (C99)
isnormal [V1.08 or later]	Determines whether a floating-point number is normal (neither zero, subnormal, infinite, nor NaN) (C99)
signbit [V1.08 or later]	Determines whether the sign of a floating-point number is negative (C99)
acos	Arc cosine
acosf	Arc cosine
acosl [V1.08 or later]	Arc cosine (C99)
asin	Arc sine
asinf	Arc sine
asinl [V1.08 or later]	Arc sine (C99)
atan	Arc tangent
atanf	Arc tangent
atanl [V1.08 or later]	Arc tangent (C99)
atan2	Arc tangent (y / x)
atan2f	Arc tangent (y / x)
atan2l [V1.08 or later]	Arc tangent (y / x) (C99)
cos	Cosine
cosf	Cosine
cosl [V1.08 or later]	Cosine (C99)
sin	Sine
sinf	Sine
sinl [V1.08 or later]	Sine (C99)
tan	Tangent
tanf	Tangent
tanl [V1.08 or later]	Tangent (C99)
acosh [V1.08 or later]	Arc hyperbolic cosine (C99)
acoshf [V1.08 or later]	Arc hyperbolic cosine (C99)
acoshl [V1.08 or later]	Arc hyperbolic cosine (C99)
asinh [V1.08 or later]	Arc hyperbolic sine (C99)

Function/Macro Name	Outline
asinhf [V1.08 or later]	Arc hyperbolic sine (C99)
asinhl [V1.08 or later]	Arc hyperbolic sine (C99)
atanh [V1.08 or later]	Arc hyperbolic tangent (C99)
atanhf [V1.08 or later]	Arc hyperbolic tangent (C99)
atanhl [V1.08 or later]	Arc hyperbolic tangent (C99)
cosh	Hyperbolic cosine
coshf	Hyperbolic cosine
coshl [V1.08 or later]	Hyperbolic cosine (C99)
sinh	Hyperbolic sine
sinhf	Hyperbolic sine
sinhl [V1.08 or later]	Hyperbolic sine (C99)
tanh	Hyperbolic tangent
tanhf	Hyperbolic tangent
tanhl [V1.08 or later]	Hyperbolic tangent (C99)
exp	Exponent function (natural logarithm)
expf	Exponent function (natural logarithm)
expl [V1.08 or later]	Exponent function (natural logarithm) (C99)
frexp	Break a floating-point number into a normalized fraction and an integral power of 2
frexpf	Break a floating-point number into a normalized fraction and an integral power of 2
frexpl [V1.08 or later]	Break a floating-point number into a normalized fraction and an integral power of 2 (C99)
ldexp	Multiply a floating-point number by an integral power of 2
ldexpf	Multiply a floating-point number by an integral power of 2
ldexpl [V1.08 or later]	Multiply a floating-point number by an integral power of 2 (C99)
log	Logarithmic function (natural logarithm)
logf	Logarithmic function (natural logarithm)
logl [V1.08 or later]	Logarithmic function (natural logarithm) (C99)
log10	Logarithmic function (base = 10)
log10f	Logarithmic function (base = 10)
log10l [V1.08 or later]	Logarithmic function (base = 10) (C99)
log1p [V1.08 or later]	Natural logarithm of 1 plus the argument (C99)
log1pf [V1.08 or later]	Natural logarithm of 1 plus the argument (C99)
log1pl [V1.08 or later]	Natural logarithm of 1 plus the argument (C99)
modf	Break a floating-point number into integral and fractional parts
modff	Break a floating-point number into integral and fractional parts
modfl [V1.08 or later]	Break a floating-point number into integral and fractional parts (C99)

Function/Macro Name	Outline
scalbn [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
scalbnf [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
scalbnl [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
scalbln [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
scalblnf [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
scalblnl [V1.09 or later]	Multiply a floating-point number by an integral power of FLT_RADIX (C99)
fabs	Absolute value function
fabsf	Absolute value function
fabsl [V1.08 or later]	Absolute value function (C99)
pow	Power function
powf	Power function
powl [V1.08 or later]	Power function (C99)
sqrt	Square root function
sqrtf	Square root function
sqrtl [V1.08 or later]	Square root function (C99)
ceil	The smallest integer value not less than a floating-point number
ceilf	The smallest integer value not less than a floating-point number
ceill [V1.08 or later]	The smallest integer value not less than a floating-point number (C99)
floor	The largest integer value not greater than a floating-point number
floorf	The largest integer value not greater than a floating-point number
floorl [V1.08 or later]	The largest integer value not greater than a floating-point number (C99)
nearbyint [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
nearbyintf [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
nearbyintl [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
rint [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
rintf [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
rintl [V1.09 or later]	Rounding to an integer value in floating-point format according to the current rounding direction (C99)
lrint [V1.09 or later]	Rounding to a long type integer value according to the current rounding direction (C99)
lrintf [V1.09 or later]	Rounding to a long type integer value according to the current rounding direction (C99)
lrintl [V1.09 or later]	Rounding to a long type integer value according to the current rounding direction (C99)

Function/Macro Name	Outline
llrint [V1.09 or later]	Rounding to a long long type integer value according to the current rounding direction (C99)
llrintf [V1.09 or later]	Rounding to a long long type integer value according to the current rounding direction (C99)
llrintl [V1.09 or later]	Rounding to a long long type integer value according to the current rounding direction (C99)
round [V1.09 or later]	Rounding to integer value in floating-point format (C99)
roundf [V1.09 or later]	Rounding to integer value in floating-point format (C99)
roundl [V1.09 or later]	Rounding to integer value in floating-point format (C99)
lround [V1.09 or later]	Rounding to a long type integer value (C99)
lroundf [V1.09 or later]	Rounding to a long type integer value (C99)
lroundl [V1.09 or later]	Rounding to a long type integer value (C99)
llround [V1.09 or later]	Rounding to a long long type integer value (C99)
llroundf [V1.09 or later]	Rounding to a long long type integer value (C99)
llroundl [V1.09 or later]	Rounding to a long long type integer value (C99)
trunc [V1.09 or later]	Rounding to truncated integer value (C99)
truncf [V1.09 or later]	Rounding to truncated integer value (C99)
truncl [V1.09 or later]	Rounding to truncated integer value (C99)
fmod	Remainder function
fmodf	Remainder function
fmodl [V1.08 or later]	Remainder function (C99)
copysign [V1.09 or later]	Generates a value consisting of the given absolute value and sign (C99)
copysignf [V1.09 or later]	Generates a value consisting of the given absolute value and sign (C99)
copysignl [V1.09 or later]	Generates a value consisting of the given absolute value and sign (C99)
nan [V1.09 or later]	Convert character string to NaN (C99)
nanf [V1.09 or later]	Convert character string to NaN (C99)
nanl [V1.09 or later]	Convert character string to NaN (C99)
fdim [V1.09 or later]	Calculation of the positive difference (C99)
fdimf [V1.09 or later]	Calculation of the positive difference (C99)
fdiml [V1.09 or later]	Calculation of the positive difference (C99)
fmax [V1.09 or later]	Obtaing the greater value (C99)
fmaxf [V1.09 or later]	Obtaing the greater value (C99)
fmaxl [V1.09 or later]	Obtaing the greater value (C99)
fmin [V1.09 or later]	Obtaing the smaller value (C99)
fminf [V1.09 or later]	Obtaing the smaller value (C99)
fminl [V1.09 or later]	Obtaing the smaller value (C99)

Function/Macro Name	Outline
isgreater [V1.09 or later]	Determining whether the first argument is greater than the second argument (C99)
isgreaterequal [V1.09 or later]	Determining whether the first argument is equal to or greater than the second argument (C99)
isless [V1.09 or later]	Determining whether the first argument is smaller than the second argument (C99)
islessequal [V1.09 or later]	Determining whether the first argument is equal to or smaller than the second argument (C99)
islessgreater [V1.09 or later]	Determining whether the first argument is smaller or greater than the second argument (C99)
isunordered [V1.09 or later]	Determining whether the arguments are not ordered (C99)

fpclassify [V1.08 or later]

Classifies its argument value as NaN, infinite, normal, subnormal or zero.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int fpclassify(real-floating *x*); (C99)

[Return value]

If the value of the argument is NaN, FP_NAN is returned.
If the value of the argument is infinite, FP_INFINITE is returned.
If the value of the argument is a normal number, FP_NORMAL is returned.
If the value of the argument is a subnormal number, FP_SUBNORMAL is returned.
If the value of the argument is 0, FP_ZERO is returned.
For the value returned by each macro, refer to math.h.

[Description]

This macro classifies its argument value as NaN, infinite, normal, subnormal or zero.
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

isfinite [V1.08 or later]

Determines whether its argument has a finite value (zero, subnormal, or normal).

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isfinite(real-floating *x*); (C99)

[Return value]

If the argument is finite (0, subnormal number, or normal number), a value other than 0 is returned.
If the argument is infinite or NaN, 0 is returned.

[Description]

This macro determines whether its argument has a finite value (zero, subnormal, or normal).
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

isinf [V1.08 or later]

Determines whether its argument value is an infinity.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isinf(real-floating *x*); (C99)

[Return value]

If the argument is positive or negative infinite, a value other than 0 is returned.
In other cases, 0 is returned.

[Description]

This macro determines whether its argument value is an infinity (positive or negative).
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

isnan [V1.08 or later]

Determines whether its argument value is a NaN.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isnan(real-floating *x*); (C99)

[Return value]

If the argument is NaN, a value other than 0 is returned.
In other cases, 0 is returned.

[Description]

This macro determines whether its argument value is a NaN.
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

isnormal [V1.08 or later]

Determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN).

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isnormal(real-floating *x*); (C99)

[Return value]

If the argument is a normal number, a value other than 0 is returned.
If the argument is 0, a subnormal number, infinite, or NaN, 0 is returned.

[Description]

This macro determines whether its argument value is normal (neither zero, subnormal, infinite, nor NaN).
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

signbit [V1.08 or later]

Determines whether the sign of its argument value is negative.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int signbit(real-floating *x*); (C99)

[Return value]

If the sign of the argument is negative, a value other than 0 is returned.
If the sign of the argument is positive, 0 is returned.

[Description]

This macro determines whether the sign of its argument value is negative.
If a type other than the floating-point type is passed to the argument, correct operation is not guaranteed.

acos

Arc cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far acos(double x);
```

[Return value]

Returns the arc cosine of *x*. The returned value is in radian and in a range of [0, π].
If *x* is not between [-1, 1], acos returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc cosine of *x*. Specify *x* as, -1<= *x* <= 1.

acosf

Arc cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   acosf(float x);
```

[Return value]

Returns the arc cosine of *x*. The returned value is in radian and in a range of [0, π].
If *x* is not between [-1, 1], acosf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc cosine of *x*. Specify *x* as, -1<= *x* <= 1.

acosl [V1.08 or later]

Arc cosine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far acosl(long double *x*); (C99)

[Return value]

Returns the arc cosine of *x*. The returned value is in radian and in a range of [0, π].
If *x* is not between [-1, 1], acosl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc cosine of *x*. Specify *x* as, -1<= *x* <= 1.

asin

Arc sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far asin(double x);
```

[Return value]

Returns the arc sine (arcsine) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
If *x* is not between [-1, 1], asin returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc sine (arcsine) of *x*. Specify *x* as, -1 <= *x* <= 1.

asinf

Arc sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   asinf(float x);
```

[Return value]

Returns the arc sine (arcsine) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
If *x* is not between [-1, 1], asinf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc sine (arcsine) of *x*. Specify *x* as, -1 <= *x* <= 1.

asinl [V1.08 or later]

Arc sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far asinl(long double x); (C99)
```

[Return value]

Returns the arc sine (arcsine) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
If *x* is not between [-1, 1], asinl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc sine (arcsine) of *x*. Specify *x* as, -1 <= *x* <= 1.

atan

Arc tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far  atan(double x);
```

[Return value]

Returns the arc tangent (arctangent) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
When *x* is Not-a-Number, this function returns Not-a-Number and sets macro EDOM to global variable errno.
When *x* is -0, this function returns -0.
If the solution is a denormal number, atan sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent (arctangent) of *x*.

atanf

Arc tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   atanf(float x);
```

[Return value]

Returns the arc tangent (arctangent) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
When *x* is Not-a-Number, this function returns Not-a-Number and sets macro EDOM to global variable errno.
When *x* is -0, this function returns -0.
If the solution is a denormal number, atanf sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent (arctangent) of *x*.

atanl [V1.08 or later]

Arc tangent

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far atanl(long double *x*); (C99)

[Return value]

Returns the arc tangent (arctangent) of *x*. The returned value is in radian and in a range of [-π / 2, π / 2].
When *x* is Not-a-Number, this function returns Not-a-Number and sets macro EDOM to global variable errno.
When *x* is -0, this function returns -0.
If the solution is a denormal number, atanl sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent (arctangent) of *x*.

atan2

Arc tangent (y / x)

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far atan2(double y, double x);
```

[Return value]

Returns the arc tangent (arctangent) of *y* / *x*. The returned value is in radian and in a range of [-π, π].

When either *x* or *y* is Not-a-Number, *x* and *y* are both 0, or *x* and *y* are both ±∞, this function returns Not-a-Number and sets macro EDOM to global variable errno.

If the solution is a denormal number or has disappeared and became 0, atan2 sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent of *y* / *x*. At this time, the quadrant of the return value is determined based on the sign of both arguments.

atan2f

Arc tangent (y / x)

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   atan2f(float y, float x);
```

[Return value]

Returns the arc tangent (arctangent) of *y* / *x*. The returned value is in radian and in a range of [-π, π].

When either *x* or *y* is Not-a-Number, *x* and *y* are both 0, or *x* and *y* are both ±∞, this function returns Not-a-Number and sets macro EDOM to global variable errno.

If the solution is a denormal number or has disappeared and became 0, atan2f sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent of *y* / *x*. At this time, the quadrant of the return value is determined based on the sign of both arguments.

atan2l [V1.08 or later]

Arc tangent (y / x)

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far atan2l(long double *y*, long double *x*); (C99)

[Return value]

Returns the arc tangent (arctangent) of *y* / *x*. The returned value is in radian and in a range of [-π, π].
When either *x* or *y* is Not-a-Number, *x* and *y* are both 0, or *x* and *y* are both ±∞, this function returns Not-a-Number and sets macro EDOM to global variable errno.
If the solution is a denormal number or has disappeared and became 0, atan2l sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent of *y* / *x*. At this time, the quadrant of the return value is determined based on the sign of both arguments.

cos

Cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far cos(double x);
```

[Return value]

Returns the cosine of *x*.
If *x* is Not-a-Number or ±∞, cos returns Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, cos sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

cosf

Cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   cosf(float x);
```

[Return value]

Returns the cosine of *x*.
If *x* is Not-a-Number or ±∞, cosf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, cosf sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

cosl [V1.08 or later]

Cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far cosl(long double x); (C99)
```

[Return value]

Returns the cosine of *x*.
If *x* is Not-a-Number or ±∞, cosl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, cosl sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

sin

Sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far  sin(double x);
```

[Return value]

Returns the sine of *x*.
If *x* is Not-a-Number or ±∞, sin returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, sin sets macro ERANGE to global variable errno.

[Description]

This function calculates the sine of *x*. Specify the angle in radian.

sinf

Sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   sinf(float x);
```

[Return value]

Returns the sine of *x*.
If *x* is Not-a-Number or ±∞, sinf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, sinf sets macro ERANGE to global variable errno.

[Description]

This function calculates the sine of *x*. Specify the angle in radian.

sinl [V1.08 or later]

Sine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far sinl(long double *x*); (C99)

[Return value]

Returns the sine of *x*.
If *x* is Not-a-Number or ±∞, sinl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, sinl sets macro ERANGE to global variable errno.

[Description]

This function calculates the sine of *x*. Specify the angle in radian.

tan

Tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far tan(double x);
```

[Return value]

Returns the tangent of x.
If *x* is Not-a-Number or ±∞, tan returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, tan sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

tanf

Tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   tanf(float x);
```

[Return value]

Returns the tangent of *x*.
If *x* is Not-a-Number or ±∞, tanf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, tanf sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

tanl [V1.08 or later]

Tangent

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far tanl(long double *x*); (C99)

[Return value]

Returns the tangent of *x*.
If *x* is Not-a-Number or ±∞, tanl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, tanl sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

acosh [V1.08 or later]

Arc hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far acosh(double x); (C99)
```

[Return value]

Returns the arc hyperbolic cosine of *x*.
If *x* is smaller than 1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc hyperbolic cosine of *x*. Specify the angle in radian.

acoshf [V1.08 or later]

Arc hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far acoshf(float x); (C99)
```

[Return value]

Returns the arc hyperbolic cosine of *x*.
If *x* is smaller than 1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc hyperbolic cosine of *x*. Specify the angle in radian.

acoshl [V1.08 or later]

Arc hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far acoshl(long double *x*); (C99)

[Return value]

Returns the arc hyperbolic cosine of *x*.
If *x* is smaller than 1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the arc hyperbolic cosine of *x*. Specify the angle in radian.

asinh [V1.08 or later]

Arc hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far asinh(double x); (C99)
```

[Return value]

Returns the arc hyperbolic sine of *x*.

[Description]

This function calculates the arc hyperbolic sine of *x*. Specify the angle in radian.

asinhf [V1.08 or later]

Arc hyperbolic sine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
float __far asinhf(float *x*); (C99)

[Return value]

Returns the arc hyperbolic sine of *x*.

[Description]

This function calculates the arc hyperbolic sine of *x*. Specify the angle in radian.

asinhl [V1.08 or later]

Arc hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far asinhl(long double x); (C99)
```

[Return value]

Returns the arc hyperbolic sine of *x*.

[Description]

This function calculates the arc hyperbolic sine of *x*. Specify the angle in radian.

atanh [V1.08 or later]

Arc hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far atanh(double x); (C99)
```

[Return value]

Returns the arc hyperbolic tangent of *x*.
If *x* is not inside the range [-1, +1], returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VAL and sets macro ERANGE to global variable errno.
If *x* is 1, returns HUGE_VAL and sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc hyperbolic tangent of *x*. Specify the angle in radian.

atanhf [V1.08 or later]

Arc hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

#include <math.h>
float __far atanhf(float *x*); (C99)

[Return value]

Returns the arc hyperbolic tangent of *x*.
If *x* is not inside the range [-1, +1], returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VALF and sets macro ERANGE to global variable errno.
If *x* is 1, returns HUGE_VALF and sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc hyperbolic tangent of *x*. Specify the angle in radian.

atanhl [V1.08 or later]

Arc hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far atanhl(long double *x*); (C99)

[Return value]

Returns the arc hyperbolic tangent of *x*.
If *x* is not inside the range [-1, +1], returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VALL and sets macro ERANGE to global variable errno.
If *x* is 1, returns HUGE_VALL and sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc hyperbolic tangent of *x*. Specify the angle in radian.

cosh

Hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far cosh(double x);
```

[Return value]

Returns the hyperbolic cosine of *x*.
cosh returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic cosine of *x*. Specify the angle in radian. The definition expression is as follows.

$$(e^{x} + e^{-x}) / 2$$

coshf

Hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   coshf(float x);
```

[Return value]

Returns the hyperbolic cosine of *x*.
coshf returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic cosine of *x*. Specify the angle in radian. The definition expression is as follows.

$$(e^{x} + e^{-x}) / 2$$

coshl [V1.08 or later]

Hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far coshl(long double *x*); (C99)

[Return value]

Returns the hyperbolic cosine of *x*.
coshl returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic cosine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} + e^{-x}) / 2$

sinh

Hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far sinh(double x);
```

[Return value]

Returns the hyperbolic sine of *x*.
sinh returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic sine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} - e^{-x}) / 2$

sinhf

Hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   sinhf(float x);
```

[Return value]

Returns the hyperbolic sine of *x*.
sinhf returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic sine of *x*. Specify the angle in radian. The definition expression is as follows.

$$(e^{x} - e^{-x}) / 2$$

sinhl [V1.08 or later]

Hyperbolic sine

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far sinhl(long double *x*); (C99)

[Return value]

Returns the hyperbolic sine of *x*.
sinhl returns ∞ and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic sine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} - e^{-x}) / 2$

tanh

Hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far tanh(double x);
```

[Return value]

Returns the hyperbolic tangent of *x*.
If the solution is a denormal number, tanh sets macro ERANGE to global variable errno.

[Description]

This function calculates the hyperbolic tangent of *x*. Specify the angle in radian. The definition expression is as follows.

sinh (*x*) / cosh (*x*)

tanhf

Hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   tanhf(float x);
```

[Return value]

Returns the hyperbolic tangent of *x*.
If the solution is a denormal number, tanhf sets macro ERANGE to global variable errno.

[Description]

This function calculates the hyperbolic tangent of *x*. Specify the angle in radian. The definition expression is as follows.

sinhf (*x*) / coshf (*x*)

tanhl [V1.08 or later]

Hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far tanhl(long double x); (C99)
```

[Return value]

Returns the hyperbolic tangent of *x*.
If the solution is a denormal number, tanhl sets macro ERANGE to global variable errno.

[Description]

This function calculates the hyperbolic tangent of *x*. Specify the angle in radian. The definition expression is as follows.

sinhl (*x*) / coshl (*x*)

exp

Obtain an exponent function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far exp(double x);
```

[Return value]

Returns the *x*th power of e.
exp returns 0 or an denormal number if an underflow occurs, and sets macro ERANGE to global variable errno.
If an overflow occurs, exp returns ∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the *x*th power of e (e is the base of a natural logarithm and is about 2.71828).

expf

Obtain an exponent function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   expf(float x);
```

[Return value]

Returns the *x*th power of e.
expf returns 0 or an denormal number if an underflow occurs, and sets macro ERANGE to global variable errno.
If an overflow occurs, expf returns ∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the *x*th power of e (e is the base of a natural logarithm and is about 2.71828).

expl [V1.08 or later]

Obtain an exponent function

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far expl(long double *x*); (C99)

[Return value]

Returns the *x*th power of e.
expl returns 0 or an denormal number if an underflow occurs, and sets macro ERANGE to global variable errno.
If an overflow occurs, expl returns ∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the *x*th power of e (e is the base of a natural logarithm and is about 2.71828).

frexp

Divide floating-point number into mantissa and power

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far frexp(double val, int *exp);
```

[Return value]

Returns the mantissa of *val*. The value to be returned ranges between [1 / 2, 1) or is 0.
frexp sets 0 to **exp* and returns 0 if *val* is 0.
If *val* is a Not-a-Number (NaN) or ±∞, frexp returns a Not-a-Number (NaN) and sets 0 to **exp* and macro EDOM to global variable errno.

[Description]

Divides a floating-point number into a normalized number and an integral power of 2. The integral power of 2 is stored in **exp*.
This function expresses *val* of double type as mantissa m and the pth power of 2. The resulting mantissa m is 0.5 <= | m | < 1.0, unless *val* is zero. p is stored in **exp*. m and p are calculated so that *val* = m * 2^{p}.

frexpf

Divide floating-point number into mantissa and power

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   frexpf(float val, int *exp);
```

[Return value]

Returns the mantissa of *val*. The value to be returned ranges between [1 / 2, 1) or is 0.
frexpf sets 0 to **exp* and returns 0 if *val* is 0.
If *val* is a Not-a-Number (NaN) or ±∞, frexpf returns a Not-a-Number (NaN) and sets 0 to **exp* and macro EDOM to global variable errno.

[Description]

Divides a floating-point number into a normalized number and an integral power of 2. The integral power of 2 is stored in **exp*.
This function expresses *val* of float type as mantissa m and the pth power of 2. The resulting mantissa m is 0.5 <= | m | < 1.0, unless *val* is zero. p is stored in **exp*. m and p are calculated so that *val* = m * 2^{p}.

frexpl [V1.08 or later]

Divide floating-point number into mantissa and power

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far frexpl(long double *val*, int **exp*); (C99)

[Return value]

Returns the mantissa of *val*. The value to be returned ranges between [1 / 2, 1) or is 0.
frexpl sets 0 to **exp* and returns 0 if *val* is 0.
If *val* is a Not-a-Number (NaN) or ±∞, frexpl returns a Not-a-Number (NaN) and sets 0 to **exp* and macro EDOM to global variable errno.

[Description]

Divides a floating-point number into a normalized number and an integral power of 2. The integral power of 2 is stored in **exp*.
This function expresses val of long double type as mantissa m and the pth power of 2. The resulting mantissa m is 0.5 <= | m | < 1.0, unless *val* is zero. p is stored in **exp*. m and p are calculated so that *val* = m * 2^{p}.

ldexp

Multiply a floating-point number and an integral power of 2

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far ldexp(double val, int exp);
```

[Return value]

Returns the value calculated by*val* x 2^{exp}.

When val is a Not-a-Number (NaN), ldexp returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

If an underflow or overflow occurs as a result of executing ldexp, it sets macro ERANGE to global variable errno. If an underflow occurs, ldexp returns an denormal number. If an overflow occurs, it returns ∞.

[Description]

Multiply a floating-point number and an integral power of 2.

ldexpf

Multiply a floating-point number and an integral power of 2

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   ldexpf(float val, int exp);
```

[Return value]

Returns the value calculated by*val* x 2^{exp}.

When val is a Not-a-Number (NaN), ldexpf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

If an underflow or overflow occurs as a result of executing ldexpf, it sets macro ERANGE to global variable errno. If an underflow occurs, ldexpf returns an denormal number. If an overflow occurs, it returns ∞.

[Description]

Multiply a floating-point number and an integral power of 2.

ldexpl [V1.08 or later]

Multiply a floating-point number and an integral power of 2

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far ldexpl(long double *val*, int *exp*); (C99)

[Return value]

Returns the value calculated by*val* x 2^{exp}.
When val is a Not-a-Number (NaN), ldexpl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If an underflow or overflow occurs as a result of executing ldexpl, it sets macro ERANGE to global variable errno. If an underflow occurs, ldexpl returns an denormal number. If an overflow occurs, it returns ∞.

[Description]

Multiply a floating-point number and an integral power of 2.

log

Obtain the natural logarithm

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far log(double x);
```

[Return value]

Returns the natural logarithm of *x*.
log returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of *x*, i.e., logarithm with base e.

logf

Obtain the natural logarithm

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   logf(float x);
```

[Return value]

Returns the natural logarithm of *x*.
logf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of *x*, i.e., logarithm with base e.

logl [V1.08 or later]

Obtain the natural logarithm

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far logl(long double x); (C99)
```

[Return value]

Returns the natural logarithm of *x*.
logl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of *x*, i.e., logarithm with base e.

log10

Obtain the logarithm with a base of 10

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far    log10(double x);
```

[Return value]

Returns the logarithm of *x* with base 10.
log10 returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the logarithm of *x* with base 10.

log10f

Obtain the logarithm with a base of 10

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   log10f(float x);
```

[Return value]

Returns the logarithm of *x* with base 10.
log10f returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the logarithm of *x* with base 10.

log10l [V1.08 or later]

Obtain the logarithm with a base of 10

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far log10l(long double *x*); (C99)

[Return value]

Returns the logarithm of *x* with base 10.
log10l returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is negative.
If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the logarithm of *x* with base 10.

log1p [V1.08 or later]

Natural logarithm of 1 plus the argument

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far log1p(double x); (C99)
```

[Return value]

Returns the natural logarithm of 1 plus *x*.
If *x* is smaller than -1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VAL and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of 1 plus *x*.

log1pf [V1.08 or later]

Natural logarithm of 1 plus the argument

[Classification]

Mathematical library

[Syntax]

#include <math.h>
float __far log1pf(float *x*); (C99)

[Return value]

Returns the natural logarithm of 1 plus *x*.
If *x* is smaller than -1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VALF and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of 1 plus *x*.

log1pl [V1.08 or later]

Natural logarithm of 1 plus the argument

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far log1pl(long double *x*); (C99)

[Return value]

Returns the natural logarithm of 1 plus *x*.
If *x* is smaller than -1, returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is -1, returns -HUGE_VALL and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of 1 plus *x*.

modf

Divide floating-point number into integer and decimal

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far modf(double val, double *iptr);
```

[Return value]

Returns a signed decimal part. The sign of the result is the same as the sign of *val*.
When *val* is ±∞, this function returns 0 and sets ±∞ to iptr and macro EDOM to global variable errno.
When *val* is Not-a-Number, this function returns Not-a-Number and sets Not-a-Number to iptr and macro EDOM to global variable errno.

[Description]

This function divides *val* into integer and decimal parts, and stores the integer part in **iptr*. The sign of the integer and decimal parts is the same as the sign of *val*.

modff

Divide floating-point number into integer and decimal

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   modff(float val, float *iptr);
```

[Return value]

Returns a signed decimal part. The sign of the result is the same as the sign of *val*.
When *val* is ±∞, this function returns 0 and sets ±∞ to iptr and macro EDOM to global variable errno.
When *val* is Not-a-Number, this function returns Not-a-Number and sets Not-a-Number to iptr and macro EDOM to global variable errno.

[Description]

This function divides *val* into integer and decimal parts, and stores the integer part in **iptr*. The sign of the integer and decimal parts is the same as the sign of *val*.

modfl [V1.08 or later]

Divide floating-point number into integer and decimal

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far modfl(long double *val*, long double **iptr*); (C99)

[Return value]

Returns a signed decimal part. The sign of the result is the same as the sign of *val*.
When *val* is ±∞, this function returns 0 and sets ±∞ to iptr and macro EDOM to global variable errno.
When *val* is Not-a-Number, this function returns Not-a-Number and sets Not-a-Number to iptr and macro EDOM to global variable errno.

[Description]

This function divides *val* into integer and decimal parts, and stores the integer part in **iptr*. The sign of the integer and decimal parts is the same as the sign of *val*.

scalbn [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
double __far scalbn(double *x*, int *n*); (C99)

[Return value]

Returns the value calculated by $x \times FLT\_RADIX^n$.
When *x* is a Not-a-Number (NaN), scalbn returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If an underflow or overflow occurs as a result of executing scalbn, it sets macro ERANGE to global variable errno. If an underflow occurs, scalbn returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

scalbnf [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far scalbnf(float x, int n); (C99)
```

[Return value]

Returns the value calculated by $x \times FLT\_RADIX^n$.

When *x* is a Not-a-Number (NaN), scalbnf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

If an underflow or overflow occurs as a result of executing scalbnf, it sets macro ERANGE to global variable errno. If an underflow occurs, scalbnf returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

scalbnl [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far scalbnl(long double *x*, int *n*); (C99)

[Return value]

Returns the value calculated by *x* x FLT_RADIX^{n}.
When *x* is a Not-a-Number (NaN), scalbnl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If an underflow or overflow occurs as a result of executing scalbnl, it sets macro ERANGE to global variable errno. If an underflow occurs, scalbnl returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

scalbln [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
double __far scalbln(double *x*, long int *n*); (C99)

[Return value]

Returns the value calculated by $x \times FLT\_RADIX^n$.

When *x* is a Not-a-Number (NaN), scalbln returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

If an underflow or overflow occurs as a result of executing scalbln, it sets macro ERANGE to global variable errno. If an underflow occurs, scalbln returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

scalblnf [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
float __far scalblnf(float *x*, long int *n*); (C99)

[Return value]

Returns the value calculated by $x \times FLT\_RADIX^n$.

When *x* is a Not-a-Number (NaN), scalblnf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

If an underflow or overflow occurs as a result of executing scalblnf, it sets macro ERANGE to global variable errno. If an underflow occurs, scalblnf returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

scalblnl [V1.09 or later]

Multiply a floating-point number by an integral power of FLT_RADIX.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far scalblnl(long double *x*, long int *n*); (C99)

[Return value]

Returns the value calculated by $x \times FLT\_RADIX^n$.
When *x* is a Not-a-Number (NaN), scalblnl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If an underflow or overflow occurs as a result of executing scalblnl, it sets macro ERANGE to global variable errno. If an underflow occurs, scalblnl returns an denormal number. If an overflow occurs, it returns ∞ .

[Description]

Multiply a floating-point number by an integral power of FLT_RADIX.

fabs

Calculates the absolute value

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far fabs(double x);
```

[Return value]

Returns the absolute value (size) of *x*.
If *x* is ±∞, fabs returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), fabs returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the absolute value (size) of *x*.

fabsf

Calculates the absolute value

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   fabsf(float x);
```

[Return value]

Returns the absolute value (size) of *x*.
If *x* is ±∞, fabsf returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), fabsf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the absolute value (size) of *x*.

fabsl [V1.08 or later]

Calculates the absolute value

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far fabsl(long double *x*); (C99)

[Return value]

Returns the absolute value (size) of *x*.
If *x* is ±∞, fabsl returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), fabsl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the absolute value (size) of *x*.

pow

Calculates the *y*th power of *x*.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far pow(double x, double y);
```

[Return value]

Returns the *y*th power of *x*.

If *x* < 0 and *y* is a non-integer or if *x* = 0 and *y* <= 0, pow returns a Not-a-Number (NaN) and sets the macro EDOM for the global variable errno.

If an overflow occurs, pow returns ±∞ and sets the macro ERANGE for errno.

If an underflow occurs, pow returns a denormal number and sets the macro ERANGE for errno.

[Description]

This function calculates the *y*th power of *x*.

powf

Calculates the *y*th power of *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   powf(float x, float y);
```

[Return value]

Returns the *y*th power of *x*.
If *x* < 0 and *y* is a non-integer or if *x* = 0 and *y* <= 0, powf returns a Not-a-Number (NaN) and sets the macro EDOM for the global variable errno.
If an overflow occurs, powf returns ±∞ and sets the macro ERANGE for errno.
If an underflow occurs, powf returns a denormal number and sets the macro ERANGE for errno.

[Description]

This function calculates the *y*th power of *x*.

powl [V1.08 or later]

Calculates the *y*th power of *x*

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far powl(long double *x*, long double *y*); (C99)

[Return value]

Returns the *y*th power of *x*.
If *x* < 0 and *y* is a non-integer or if *x* = 0 and *y* <= 0, powl returns a Not-a-Number (NaN) and sets the macro EDOM for the global variable errno.
If an overflow occurs, powl returns ±∞ and sets the macro ERANGE for errno.
If an underflow occurs, powl returns a denormal number and sets the macro ERANGE for errno.

[Description]

This function calculates the *y*th power of *x*.

sqrt

Calculates the square root

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far  sqrt(double x);
```

[Return value]

Returns the square root of *x*.
sqrt returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is a negative number.

[Description]

This function calculates the square root of *x*.

sqrtf

Calculates the square root

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   sqrtf(float x);
```

[Return value]

Returns the square root of *x*.
sqrtf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is a negative number.

[Description]

This function calculates the square root of *x*.

sqrtl [V1.08 or later]

Calculates the square root

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far sqrtl(long double *x*); (C99)

[Return value]

Returns the square root of *x*.
sqrtl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno if *x* is a negative number.

[Description]

This function calculates the square root of *x*.

ceil

Calculates the minimum integer value greater than *x* and *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far  ceil(double x);
```

[Return value]

Returns the minimum integer greater than *x* and *x*.
If *x* is ±∞, ceil returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), ceil returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the minimum integer value greater than *x* and *x*.

ceilf

Calculates the minimum integer value greater than *x* and *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   ceilf(float x);
```

[Return value]

Returns the minimum integer greater than *x* and *x*.
If *x* is ±∞, ceilf returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), ceilf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the minimum integer value greater than *x* and *x*.

ceill [V1.08 or later]

Calculates the minimum integer value greater than *x* and *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far ceill(long double x); (C99)
```

[Return value]

Returns the minimum integer greater than *x* and *x*.
If *x* is ±∞, ceill returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), ceill returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the minimum integer value greater than *x* and *x*.

floor

Calculates the maximum integer value less than *x* and *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far floor(double x);
```

[Return value]

Returns the maximum integer value less than *x* and *x*.
If *x* is ±∞, floor returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), floor returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the maximum integer value less than *x* and *x*.

floorf

Calculates the maximum integer value less than *x* and *x*

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   floorf(float x);
```

[Return value]

Returns the maximum integer value less than *x* and *x*.
If *x* is ±∞, floorf returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), floorf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the maximum integer value less than *x* and *x*.

floorl [V1.08 or later]

Calculates the maximum integer value less than *x* and *x*

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far floorl(long double *x*); (C99)

[Return value]

Returns the maximum integer value less than *x* and *x*.
If *x* is ±∞, floorl returns ±∞ and sets macro ERANGE to global variable errno.
If *x* is a Not-a-Number(NaN), floorl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the maximum integer value less than *x* and *x*.

nearbyint [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far nearbyint(double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

nearbyintf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far nearbyintf(float x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

nearbyintl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far nearbyintl(long double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

rint [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far rint(double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction. An "inexact" floating-point exception is not generated.

rintf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far rintf(float x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction. An "inexact" floating-point exception is not generated.

rintl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far rintl(long double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction. An "inexact" floating-point exception is not generated.

lrint [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lrint(double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

lrintf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lrintf(float x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

lrintl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lrintl(long double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

llrint [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llrint(double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

llrintf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llrintf(float x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

llrintl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llrintl(long double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction.

round [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far round(double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

roundf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far roundf(float x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

roundl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far roundl(long double x); (C99)
```

[Return value]

Returns a rounded value.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

lround [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lround(double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

lroundf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lroundf(float x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

lroundl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long int __far lroundl(long double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

llround [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llround(double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

llroundf [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llroundf(float x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

llroundl [V1.09 or later]

This function rounds the argument to an integer value in a floating-point format according to the rounding direction.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long long int __far llroundl(long double x); (C99)
```

[Return value]

Returns a rounded value. If the rounded value cannot be represented in the return type, this function returns 0 and specifies the macro ERANGE for global variable errno.

[Description]

This function rounds the argument to the integer value in a floating-point format according to the rounding direction, with halfway cases rounded away from 0.

trunc [V1.09 or later]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far trunc(double x); (C99)
```

[Return value]

Returns a truncated integer value.

[Description]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

truncf [V1.09 or later]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far truncf(float x); (C99)
```

[Return value]

Returns a truncated integer value.

[Description]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

truncl [V1.09 or later]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far truncl(long double x); (C99)
```

[Return value]

Returns a truncated integer value.

[Description]

This function rounds the argument by truncating the decimal part to form an integer value in a floating-point format.

fmod

Calculate the remainder

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far fmod(double x, double y);
```

[Return value]

Returns a floating-point value that is the remainder resulting from dividing *x* by *y*.
If *x* or *y* is a Not-a-Number (NaN) , fmod returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is ±∞ or *y* is zero, fmod returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is not ±∞ or *y* is ±∞, fmod returns *x* and sets macro EDOM to global variable errno.

[Description]

This function calculates a floating-point value that is the remainder resulting from dividing x by y. In other words, the value of "*x* - i * *y*" for integer i is calculated when *y* is not 0. The result has the same sign as that of *x*, and the absolute value is smaller than that of *y*.

fmodf

Calculate the remainder

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   __far   fmodf(float x, float y);
```

[Return value]

Returns a floating-point value that is the remainder resulting from dividing *x* by *y*.
If *x* or *y* is a Not-a-Number (NaN) , fmodf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is ±∞ or *y* is zero, fmodf returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is not ±∞ or *y* is ±∞, fmodf returns *x* and sets macro EDOM to global variable errno.

[Description]

This function calculates a floating-point value that is the remainder resulting from dividing *x* by *y*. In other words, the value of "*x* - i * *y*" for integer i is calculated when *y* is not 0. The result has the same sign as that of *x*, and the absolute value is smaller than that of *y*.

fmodl [V1.08 or later]

Calculate the remainder

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far fmodl(long double *x*, long double *y*); (C99)

[Return value]

Returns a floating-point value that is the remainder resulting from dividing *x* by *y*.
If *x* or *y* is a Not-a-Number (NaN) , fmodl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is ±∞ or *y* is zero, fmodl returns a Not-a-Number (NaN) and sets macro EDOM to global variable errno.
If *x* is not ±∞ or *y* is ±∞, fmodl returns *x* and sets macro EDOM to global variable errno.

[Description]

This function calculates a floating-point value that is the remainder resulting from dividing *x* by *y*. In other words, the value of "*x* - i * *y*" for integer i is calculated when *y* is not 0. The result has the same sign as that of *x*, and the absolute value is smaller than that of *y*.

copysign [V1.09 or later]

This function generates a value that has the absolute value of *x* and the sign of *y*.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far copysign(double x, double y); (C99)
```

[Return value]

Returns a value that has the absolute value of *x* and the sign of *y*.

[Description]

This function generates a value that has the absolute value of *x* and the sign of *y*.

copysignf [V1.09 or later]

This function generates a value that has the absolute value of *x* and the sign of *y*.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far copysignf(float x, float y); (C99)
```

[Return value]

Returns a value that has the absolute value of *x* and the sign of *y*.

[Description]

This function generates a value that has the absolute value of *x* and the sign of *y*.

copysignl [V1.09 or later]

This function generates a value that has the absolute value of *x* and the sign of *y*.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far copysignl(long double x, long double y); (C99)
```

[Return value]

Returns a value that has the absolute value of *x* and the sign of *y*.

[Description]

This function generates a value that has the absolute value of *x* and the sign of *y*.

nan [V1.09 or later]

Returns a Not-a-Number (NaN).

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far nan(const char __far *tagp); (C99)
```

[Return value]

Returns a Not-a-Number (NaN).

[Description]

This function is equivalent to strtod("NAN(n character string)", (char __near * __near *)NULL).

nanf [V1.09 or later]

Returns a Not-a-Number (NaN).

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far nanf(const char __far *tagp); (C99)
```

[Return value]

Returns a Not-a-Number (NaN).

[Description]

This function is equivalent to strtof("NAN(n character string)", (char __near * __near *)NULL).

nanl [V1.09 or later]

Returns a Not-a-Number (NaN).

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far nanl(const char __far *tagp); (C99)
```

[Return value]

Returns a Not-a-Number (NaN).

[Description]

This function is equivalent to strtold("NAN(n character string)", (char __near * __near *)NULL).

fdim [V1.09 or later]

This function determines the positive difference between two arguments.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
double __far fdim(double *x*, double *y*); (C99)

[Return value]

Returns '*x* - *y*' for '*x* > *y*' and +0 for '*x* <= *y*'. Returns a NaN when *x* or *y* is a NaN.

[Description]

This function determines the positive difference between two arguments.

fdimf [V1.09 or later]

This function determines the positive difference between two arguments.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
float __far fdimf(float *x*, float *y*); (C99)

[Return value]

Returns '*x* - *y*' for '*x* > *y*' and +0 for '*x* <= *y*'. Returns a NaN when *x* or *y* is a NaN.

[Description]

This function determines the positive difference between two arguments.

fdiml [V1.09 or later]

This function determines the positive difference between two arguments.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far fdiml(long double *x*, long double *y*); (C99)

[Return value]

Returns '*x* - *y*' for '*x* > *y*' and +0 for '*x* <= *y*'. Returns a NaN when *x* or *y* is a NaN.

[Description]

This function determines the positive difference between two arguments.

fmax [V1.09 or later]

This function determines the larger value of two arguments.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
double __far fmax(double *x*, double *y*); (C99)

[Return value]

Returns the larger value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the larger value of two arguments.

fmaxf [V1.09 or later]

This function determines the larger value of two arguments.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far fmaxf(float x, float y); (C99)
```

[Return value]

Returns the larger value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the larger value of two arguments.

fmaxl [V1.09 or later]

This function determines the larger value of two arguments.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
long double __far fmaxl(long double *x*, long double *y*); (C99)

[Return value]

Returns the larger value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the larger value of two arguments.

fmin [V1.09 or later]

This function determines the smaller value of two arguments.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double __far fmin(double x, double y); (C99)
```

[Return value]

Returns the smaller value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the smaller value of two arguments.

fminf [V1.09 or later]

This function determines the smaller value of two arguments.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float __far fminf(float x, float y); (C99)
```

[Return value]

Returns the smaller value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the smaller value of two arguments.

fminl [V1.09 or later]

This function determines the smaller value of two arguments.

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
long double __far fminl(long double x, long double y); (C99)
```

[Return value]

Returns the smaller value of two arguments. If one argument is a NaN and the other is not, the latter is returned.

[Description]

This function determines the smaller value of two arguments.

isgreater [V1.09 or later]

This function determines whether the first argument is greater than the second argument.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isgreater(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns the result of '(*x*) > (*y*)'.

[Description]

This function determines whether the first argument is greater than the second argument.

isgreaterequal [V1.09 or later]

This function determines whether the first argument is greater than or equal to the second argument.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isgreaterequal(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns the result of '(*x*) >= (*y*)'.

[Description]

This function determines whether the first argument is greater than or equal to the second argument.

isless [V1.09 or later]

This function determines whether the first argument is less than the second argument.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isless(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns the result of '(*x*) < (*y*)'.

[Description]

This function determines whether the first argument is less than the second argument.

islessequal [V1.09 or later]

This function determines whether the first argument is less than or equal to the second argument.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int islessequal(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns the result of '(*x*) <= (*y*)'.

[Description]

This function determines whether the first argument is less than or equal to the second argument.

islessgreater [V1.09 or later]

This function determines whether the first argument is less than or greater than the second argument.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int islessgreater(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns the result of '(*x*) > (*y*) || (*x*) < (*y*)'.

[Description]

This function determines whether the first argument is less than or greater than the second argument.

isunordered [V1.09 or later]

This function determines whether the arguments are unordered.

[Classification]

Mathematical library

[Syntax]

#include <math.h>
int isunordered(real-floating *x*, real-floating *y*); (C99)

[Return value]

Returns 1 when the arguments are unordered and 0 otherwise.

[Description]

This function determines whether the arguments are unordered.

7.5.5 Non-local jump functions

Non-local jump functions are as follows.

Table 7.7 Non-Local Jump Functions

Function/Macro Name	Outline
setjmp	Save the environment in which a function call was made
longjmp	Restore the environment in which a function call was made

setjmp

Save the environment in which a function call was made

[Classification]

Standard library

[Syntax]

```
#include <setjmp.h>
typedef int     __near  jmp_buf[3];
int     __far   setjmp(jmp_buf env);
```

[Return value]

Calling setjmp returns 0. When longjmp is used for a non-local jump, the return value is in the second parameter, val. However, 1 is returned if *val* is 0.

[Description]

This function sets *env* as the destination for a non-local jump. In addition, the environment in which setjmp was run is saved to *env*.

[Caution]

Do not call the setjmp function indirectly using a pointer.

longjmp

Restore the environment in which a function call was made

[Classification]

Standard library

[Syntax]

```
#include <setjmp.h>
typedef int     __near  jmp_buf[3];
void    __far   longjmp(jmp_buf env, int val);
```

[Description]

This function performs a non-local jump to the place immediately after setjmp using *env* saved by setjmp.

7.5.6 Variable arguments of functions

Variable arguments of functions are as follows.

Table 7.8 Variable Arguments of Functions

Function/Macro Name	Outline
va_start	Initialization of variable for scanning argument list
va_arg	Moving variable for scanning argument list
va_copy [V1.09 or later]	Copying variable for scanning argument list (C99)
va_end	End of scanning argument list

va_start

Initialization of variable for scanning argument list

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
typedef char    __near  *va_list;
void    va_start(va_list ap, last-named-argument);
```

[Description]

This function initializes variable *ap* so that it indicates the beginning (argument next to last-named-argument) of the list of the variable arguments.

To define function *func* having a variable arguments in a portable form, the following format is used.

```
#include    <stdarg.h>

void func(arg-declarations, ...) {
        va_list ap;
        type    argN;

        va_start(ap, last-named-argument);
        argN = va_arg(ap, type);
        va_end(ap);
}
```

Remark *arg-declarations* is an argument list with the *last-named-argument* declared at the end. ", ..." that follows indicates a list of the variable arguments. va_listis the type of the variable (*ap* in the above example) used to scan the argument list.

va_arg

Moving variable for scanning argument list

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
typedef char    __near  *va_list;
type    va_arg(va_list ap, type);
```

[Description]

This function returns the argument indicated by variable *ap*, and advances variable *ap* to indicate the next argument. For the *type* of va_arg, specify the type converted when the argument is passed to the function. Although a different type can be specified for each argument, stipulate "which type of argument is passed" according to the conventions between the called function and calling function.

Since a variable argument is converted according to default argument promotions, the type after conversion should be specified. If the argument is a pointer with a constant value, a cast must be attached to the argument to clearly show that it is a pointer. For details on default argument promotions, see "(7) Default argument promotions".

In addition, the "number of arguments that are actually passed" is determined according to the conventions between the called function and calling function.

va_copy [V1.09 or later]

Copying variable for scanning argument list.

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
typedef char __near *va_list;
void va_copy(va_list dest, va_list src); (C99)
```

[Description]

This function initializes parameter *dest* as a copy of parameter *src*.

va_end

End of scanning argument list

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
typedef char    __near  *va_list;
void    va_end(va_list ap);
```

[Description]

This function indicates the end of scanning the argument list. By enclosing va_arg between va_start and va_end, scanning the list can be repeated.

7.5.7 Standard I/O functions

Standard I/O functions are as follows.

Table 7.9 Standard I/O Functions

Function/Macro Name	Outline
printf	Write text in specified format to SFR
scanf	Read text in specified format from SFR
snprintf [V1.07 or later]	Write text in specified format to array (C99)
sprintf	Write text in specified format to array
sscanf	Read text in specified format from character string
vprintf	Write text in specified format to SFR
vscanf [V1.08 or later]	Read text in specified format from SFR (C99)
vsnprintf [V1.07 or later]	Write text in specified format to array (C99)
vsprintf	Write text in specified format to array
vsscanf [V1.08 or later]	Read text in specified format from character string (C99)
getchar	Read one character from SFR
gets	Read character string from SFR
putchar	Write one character to SFR
puts	Write character string to SFR
perror	Error processing

Specifications are implemented, so that in each function, input from stdin is performed through the getchar function and output to stdout is performed through the putchar function.

To change stdin and stdout, replace the getchar function and putchar function, respectively.

Output to stderr in the perror function is the same as output to stdout, and so it is performed through the putchar function. Note that replacing the putchar function will also change stderr. To change the output destination of stderr to something other than stdout, replace the perror function

printf

Write text in specified format to SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far printf(const char __far *format, ...); (C99)
int __far printf(const char __far * restrict format, ...); (C99) [V1.07 or later]
int __far printf_tiny(const char __far *format, ...);（C90）
int __far printf_tiny(const char __far * restrict format, ...); (C99) [V1.07 or later]
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

This function converts the arguments following *format* to match the output format, and outputs them to SFR using the putchar function. The conversion method in this case complies with the format specified by the string indicated by *format*.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear.
For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99) [V1.07 or later]
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99) [V1.07 or later]
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99) [V1.07 or later]
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99) [V1.07 or later]
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries. [V1.07 or later]

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99) [V1.07 or later]
e, E	Convert a double type (float type in a single-precision function) argument to [-]d.ddddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.
p	Output the value of the pointer. The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

printf_tiny is a simplified version of printf.

When macro __PRINTF_TINY__ is defined before the -D option or stdio.h is included, the function call of printf is replaced with printf_tiny. The following restrictions apply to conversion specifications of printf_tiny.

(1) Flag
-, +, or space cannot be specified.

(2) Field width
A negative field width "*" cannot be specified.

(3) Precision
Cannot be specified.

(4) Size
ll, j, z, t, or L cannot be specified.

(5) Type specification character
f, F, e, E, g, or G cannot be specified.

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

scanf

Read text in specified format from SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far  scanf(const char __far *format, ...); (C90)
int __far  scanf(const char __far * restrict format, ...); (C99) [V1.08 or later]
```

[Return value]

The number of input fields for which scanning, conversion, and storage were executed normally is returned.
The return value does not include scanned fields that were not stored.
If an attempt is made to read to the end of the file, the return value is EOF. If no field was stored, the return value is 0.

[Description]

This function converts the input from SFR which uses the getchar function and assigns the conversion result to the object indicated by the argument following *format*. The conversion method used here complies with the format specified by the string indicated by *format*. If an input character which conflicts with the directive terminates conversion, that conflicting input character will be discarded.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives, and the directives in the format are executed in sequence. If there is no input character or execution of a directive fails due to an incorrect input, processing is terminated.

The *format* consists of the following three types of directives:

One or more Space characters	Space ( ), tab (\t), or new-line (\n). Reading of input data is executed up to immediately before the first non-white-space character (this character is left but not read) or until reading can no longer be performed.
Ordinary characters	All ASCII characters other than "%". Reading is executed by reading the next character.
Conversion specification	Fetches 0 or more arguments and directs the conversion.

Each conversion specification starts with "%". The following appear after the "%":

%[assignment-suppression-character][field-width][size][type-specification-character]

Each conversion specification is explained below.

(1) Assignment suppression character
The assignment suppression character "*" suppresses assignment of the input field.

(2) field width

This is a positive decimal integer that defines the maximum field width. When 0 is specified, there are no regulations.

It specifies the maximum number of characters that are read before the input field is converted. If the input field is smaller than this field width, scanf reads all the characters in the field and then proceeds to the next field and its conversion specification.

If a space character or a character that cannot be converted is found before the number of characters equivalent to the field width is read, the characters up to the white space or the character that cannot be converted are read and stored. Then, scanf proceeds to the next conversion specification.

(3) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.

If there is no specification, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to an int or unsigned int argument. Furthermore, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to an int pointer.

When hh is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a signed char or unsigned char argument. hh also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a signed char argument. (C99) [V1.08 or later]

When h is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a short int or unsigned short int argument. h also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a short int argument.

When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long or unsigned long argument. When l is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a double argument, and an n type specification is forcibly applied to a long pointer.

When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.

When j is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a intmax_t or uintmax_t argument, and an n type specification is forcibly applied to an intmax_t pointer. (C99) [V1.08 or later]

When z is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a size_t or signed int argument, and an n type specification is forcibly applied to an signed int pointer. (C99) [V1.08 or later]

When t is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a ptrdiff_t or unsigned int argument, and an n type specification is forcibly applied to an ptrdiff_t pointer. (C99) [V1.08 or later]

When L is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a long double argument. However, the double type and long double type have the same format in this compiler.

(4) type specification character

These are characters that specify the type of conversion that is to be applied.

The characters that specify conversion types and their meanings are as follows.

d	Read a decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
i	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
o	Read an octal integer into the corresponding argument. The corresponding type is in accordance with the size character.
u	Read an unsigned decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
x, X	Read a hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
e, f, g, E, F, G	Read a floating-point number, infinite value, or Not-a-Number (NaN) into the corresponding argument. The corresponding type is in accordance with the size character.
s	Read a string into a given array. The corresponding argument should be "char __far *arg*[ ]". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.

[ ]	Read a non-empty string into the memory area starting with argument *arg*. This area must be large enough to accommodate the string and the null character (\0) that is automatically appended to indicate the end of the string. The corresponding argument should be "char **arg*". The character pattern enclosed by [ ] can be used in place of the type specification character s. The character pattern is a character set that defines the search set of the characters constituting the input field of sscanf. If the first character within [ ] is "^", the search set is complemented, and all ASCII characters other than the characters within [ ] are included. In addition, a range specification feature that can be used as a shortcut is also available. For example, %[0-9] matches all decimal numbers. In this set, "-" cannot be specified as the first or last character. The character preceding "-" must be less in lexical sequence than the succeeding character. - %[abcd] Matches character strings that include only a, b, c, and d. - %[^abcd] Matches character strings that include any characters other than a, b, c, and d. - %[A-DW-Z] Matches character strings that include A, B, C, D, W, X, Y, and Z. - %[z-a] Matches z, -, and a (this is not considered a range specification).
c	Scan one character. The corresponding argument should be "char __far **arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
p	Store the pointer that was scanned. The corresponding argument should be "void __far ***arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Input data is not read. The number of characters that have been read so far is written to the corresponding parameter. Even though the %n directive is executed, the number of input items that are returned when the function ends is not increased. The corresponding type is in accordance with the size character.
%	Match the character "%". No conversion or assignment is performed. The conversion specification is "%%".

F conversion can only be specified for C99 libraries.

Make sure that a floating-point number (type specification characters e, f, g, E, F, and G) corresponds to thefollowing general format.

[+ | -] *ddddd* [.] *ddd* [E | e [+ | -] *ddd*]

However, the portions enclosed by [ ] in the above format are arbitrarily selected, and ddd indicates a decimal digit.

[Caution]

- scanf may stop scanning a specific field before the normal end-of-field character is reached or may stop completely.
- scanf stops scanning and storing a field and moves to the next field under the following conditions.
  - The substitution suppression character (*) appears after "%" in the format specification, and the input field at that point has been scanned but not stored.
  - A field width (positive decimal integer) specification character was read.
  - The character to be read next cannot be converted according to the conversion specification (for example, if Z is read when the specification is a decimal number).
  - The next character in the input field does not appear in the search set (or appears in the complement search set).

If scanf stops scanning the input field at that point because of any of the above reasons, it is assumed that the next character has not yet been read, and this character is used as the first character of the next field or the first character for the read operation to be executed after the input.

- scanf ends under the following conditions:
  - The next character in the input field does not match the corresponding ordinary character in the string to be converted.
  - The next character in the input field is EOF.
  - The string to be converted ends.
- If a list of characters that is not part of the conversion specification is included in the string to be converted, make sure that the same list of characters does not appear in the input. sscanf scans matching characters but does not store them. If there was a mismatch, the first character that does not match remains in the input as if it were not read.

[Restrictions]

a or A conversion and hexadecimal floating-point numbers are not supported.

snprintf [V1.07 or later]

Write text in specified format to array

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far snprintf(char __far * restrict s, size_t n, const char __far * restrict format, ...); (C99)
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

This function converts the arguments following *format* into the output format and writes them into the array indicated by *s*. The conversion method in this case complies with the format specified by the string indicated by *format*. When *n* is 0, no text is written and *s* may be a null pointer. In other cases, output characters subsequent to the (*n*-1)th character are discarded without being written to the array and the null character is written after the character string that was actually written to the array. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear.
For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99)
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99)
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99)
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99)
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries.

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99)
e, E	Convert a double type (float type in a single-precision function) argument to [-]d.ddddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99)
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99)
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.
p	Output the value of the pointer. The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

sprintf

Write text in specified format to array

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far sprintf(char __far *s, const char __far *format, ...); (C90)
int __far sprintf(char __far * restrict s, const char __far * restrict format, ...); (C99) [V1.07 or later]
int __far sprintf_tiny(char __far *s, const char __far *format, ...); (C90)
int __far sprintf_tiny(char __far * restrict s, const char __far * restrict format, ...); (C99) [V1.07 or later]
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

This function converts the arguments following *format* into the output format and writes them into the array indicated by *s*. The conversion method in this case complies with the format specified by the string indicated by *format*. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear. For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99) [V1.07 or later]
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99) [V1.07 or later]
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99) [V1.07 or later]
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99) [V1.07 or later]
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries. [V1.07 or later]

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99) [V1.07 or later]
e, E	Convert a double type (float type in a single-precision function) argument to [-]d.ddddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.
p	Output the value of the pointer. The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

sprintf_tiny is a simplified version of sprintf.

When macro __PRINTF_TINY__ is defined before the -D option or stdio.h is included, the function call of sprintf is replaced with sprintf_tiny. The following restrictions apply to conversion specifications of sprintf_tiny.

(1) Flag
-, +, or space cannot be specified.

(2) Field width
A negative field width "*" cannot be specified.

(3) Precision
Cannot be specified.

(4) Size
ll, j, z, t, or L cannot be specified.

(5) Type specification character
f, F, e, E, g, or G cannot be specified.

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

sscanf

Read text in specified format from character string

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far  sscanf(const char __far *s, const char __far *format, ...); (C90)
int __far  sscanf(const char __far * restrict s, const char __far * restrict format, ...); (C99) [V1.08 or later]
```

[Return value]

The number of input fields for which scanning, conversion, and storage were executed normally is returned. The return value does not include scanned fields that were not stored.
If an attempt is made to read to the end of the file, the return value is EOF.
If no field was stored, the return value is 0.

[Description]

This function converts the input from the string indicated by *s*, and assigns the conversion result to the object indicated by the argument following *format*. The conversion method in this case complies with the format specified by the string indicated by *format*. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives, and the directives in the format are executed in sequence. If there is no input character or execution of a directive fails due to an incorrect input, processing is terminated.

The *format* consists of the following three types of directives:

One or more Space characters	Space ( ), tab (\t), or new-line (\n). Reading of input data is executed up to immediately before the first non-white-space character (this character is left but not read) or until reading can no longer be performed.
Ordinary characters	All ASCII characters other than "%". Reading is executed by reading the next character.
Conversion specification	Fetches 0 or more arguments and directs the conversion.

Each conversion specification starts with "%". The following appear after the "%":

%[assignment-suppression-character][field-width][size][type-specification-character]

Each conversion specification is explained below.

(1) Assignment suppression character
The assignment suppression character "*" suppresses assignment of the input field.

(2) field width

This is a positive decimal integer that defines the maximum field width. When 0 is specified, there are no regulations.

It specifies the maximum number of characters that are read before the input field is converted. If the input field is smaller than this field width, sscanf reads all the characters in the field and then proceeds to the next field and its conversion specification.

If a space character or a character that cannot be converted is found before the number of characters equivalent to the field width is read, the characters up to the white space or the character that cannot be converted are read and stored. Then, sscanf proceeds to the next conversion specification.

(3) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.

If there is no specification, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to an int or unsigned int argument. Furthermore, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to an int pointer.

When hh is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a signed char or unsigned char argument. hh also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a signed char argument. (C99) [V1.08 or later]

When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a short int or unsigned short int argument. When h is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to a short int pointer.

When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long or unsigned long argument. When l is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a double argument, and an n type specification is forcibly applied to a long pointer.

When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.

When j is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a intmax_t or uintmax_t argument, and an n type specification is forcibly applied to an intmax_t pointer. (C99) [V1.08 or later]

When z is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a size_t or signed int argument, and an n type specification is forcibly applied to an signed int pointer. (C99) [V1.08 or later]

When t is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a ptrdiff_t or unsigned int argument, and an n type specification is forcibly applied to an ptrdiff_t pointer. (C99) [V1.08 or later]

When L is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a long double argument. However, the double type and long double type have the same format in this compiler.

(4) type specification character

These are characters that specify the type of conversion that is to be applied.

The characters that specify conversion types and their meanings are as follows.

d	Read a decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
i	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
o	Read an octal integer into the corresponding argument. The corresponding type is in accordance with the size character.
u	Read an unsigned decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
x, X	Read a hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
e, f, g, E, F, G	Read a floating-point number, infinite value, or Not-a-Number (NaN) into the corresponding argument. The corresponding type is in accordance with the size character.
s	Read a string into a given array. The corresponding argument should be "char __far *arg*[ ]". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.

[ ]	Read a non-empty string into the memory area starting with argument *arg*. This area must be large enough to accommodate the string and the null character (\0) that is automatically appended to indicate the end of the string. The corresponding argument should be "char **arg*". The character pattern enclosed by [ ] can be used in place of the type specification character s. The character pattern is a character set that defines the search set of the characters constituting the input field of sscanf. If the first character within [ ] is "^", the search set is complemented, and all ASCII characters other than the characters within [ ] are included. In addition, a range specification feature that can be used as a shortcut is also available. For example, %[0-9] matches all decimal numbers. In this set, "-" cannot be specified as the first or last character. The character preceding "-" must be less in lexical sequence than the succeeding character. - %[abcd] Matches character strings that include only a, b, c, and d. - %[^abcd] Matches character strings that include any characters other than a, b, c, and d. - %[A-DW-Z] Matches character strings that include A, B, C, D, W, X, Y, and Z. - %[z-a] Matches z, -, and a (this is not considered a range specification).
c	Scan one character. The corresponding argument should be "char __far **arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
p	Store the pointer that was scanned. The corresponding argument should be "void __far ***arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Input data is not read. The number of characters that have been read so far is written to the corresponding parameter. Even though the %n directive is executed, the number of input items that are returned when the function ends is not increased. The corresponding type is in accordance with the size character.
%	Match the character "%". No conversion or assignment is performed. The conversion specification is "%%".

F conversion can only be specified for C99 libraries.

Make sure that a floating-point number (type specification characters e, f, g, E, F, and G) corresponds to thefollowing general format.

[+ | -] *ddddd* [.] *ddd* [E | e [+ | -] *ddd*]

However, the portions enclosed by [ ] in the above format are arbitrarily selected, and ddd indicates a decimal digit.

[Caution]

- sscanf may stop scanning a specific field before the normal end-of-field character is reached or may stop completely.
- sscanf stops scanning and storing a field and moves to the next field under the following conditions.
  - The substitution suppression character (*) appears after "%" in the format specification, and the input field at that point has been scanned but not stored.
  - A field width (positive decimal integer) specification character was read.
  - The character to be read next cannot be converted according to the conversion specification (for example, if Z is read when the specification is a decimal number).
  - The next character in the input field does not appear in the search set (or appears in the complement search set).

If sscanf stops scanning the input field at that point because of any of the above reasons, it is assumed that the next character has not yet been read, and this character is used as the first character of the next field or the first character for the read operation to be executed after the input.

- sscanf ends under the following conditions:
  - The next character in the input field does not match the corresponding ordinary character in the string to be converted.
  - The next character in the input field is EOF.
  - The string to be converted ends.
- If a list of characters that is not part of the conversion specification is included in the string to be converted, make sure that the same list of characters does not appear in the input. sscanf scans matching characters but does not store them. If there was a mismatch, the first character that does not match remains in the input as if it were not read.

[Restrictions]

a or A conversion and hexadecimal floating-point numbers are not supported.

vprintf

Write text in specified format to SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far vprintf(const char __far *format, va_list arg); (C90)
int __far vprintf(const char __far * restrict format, va_list arg); (C99) [V1.07 or later]
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

From among the parameter sequence, this function outputs the parameter indicated by pointer *arg* to SFR, using the putchar function. The conversion method used here complies with the format specified by the string indicated by *format*.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear. For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99) [V1.07 or later]
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99) [V1.07 or later]
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99) [V1.07 or later]
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99) [V1.07 or later]
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries. [V1.07 or later]

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99) [V1.07 or later]
e, E	Convert a double type (float type in a single-precision function) argument to [-]d.ddddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer . Correct operation is not guaranteed when a null pointer is passed.
p	The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

vscanf [V1.08 or later]

Read text in specified format from SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far  vscanf(const char __far * restrict format, va_list arg); (C99) [V1.08 or later]
```

[Return value]

The number of input fields for which scanning, conversion, and storage were executed normally is returned.
The return value does not include scanned fields that were not stored.
If an attempt is made to read to the end of the file, the return value is EOF. If no field was stored, the return value is 0.

[Description]

This function converts the input from SFR which uses the getchar function and assigns the conversion result to the parameter indicated by pointer *arg* among the parameter sequence. The conversion method used here complies with the format specified by the string indicated by *format*. If an input character which conflicts with the directive terminates conversion, that conflicting input character will be discarded.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives, and the directives in the format are executed in sequence. If there is no input character or execution of a directive fails due to an incorrect input, processing is terminated.

The *format* consists of the following three types of directives:

One or more Space characters	Space ( ), tab (\t), or new-line (\n). Reading of input data is executed up to immediately before the first non-white-space character (this character is left but not read) or until reading can no longer be performed.
Ordinary characters	All ASCII characters other than "%". Reading is executed by reading the next character.
Conversion specification	Fetches 0 or more arguments and directs the conversion.

Each conversion specification starts with "%". The following appear after the "%":

%[assignment-suppression-character][field-width][size][type-specification-character]

Each conversion specification is explained below.

(1) Assignment suppression character
The assignment suppression character "*" suppresses assignment of the input field.

(2) field width

This is a positive decimal integer that defines the maximum field width. When 0 is specified, there are no regulations.

It specifies the maximum number of characters that are read before the input field is converted. If the input field is smaller than this field width, scanf reads all the characters in the field and then proceeds to the next field and its conversion specification.

If a space character or a character that cannot be converted is found before the number of characters equivalent to the field width is read, the characters up to the white space or the character that cannot be converted are read and stored. Then, scanf proceeds to the next conversion specification.

(3) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.

If there is no specification, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to an int or unsigned int argument. Furthermore, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to an int pointer.

When hh is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a signed char or unsigned char argument. hh also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a signed char argument. (C99) [V1.08 or later]

When h is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a short int or unsigned short int argument. h also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a short int argument.

When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long or unsigned long argument. When l is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a double argument, and an n type specification is forcibly applied to a long pointer.

When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.

When j is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a intmax_t or uintmax_t argument, and an n type specification is forcibly applied to an intmax_t pointer. (C99) [V1.08 or later]

When z is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a size_t or signed int argument, and an n type specification is forcibly applied to an signed int pointer. (C99) [V1.08 or later]

When t is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a ptrdiff_t or unsigned int argument, and an n type specification is forcibly applied to an ptrdiff_t pointer. (C99) [V1.08 or later]

When L is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a long double argument. However, the double type and long double type have the same format in this compiler.

(4) type specification character

These are characters that specify the type of conversion that is to be applied.

The characters that specify conversion types and their meanings are as follows.

d	Read a decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
i	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
o	Read an octal integer into the corresponding argument. The corresponding type is in accordance with the size character.
u	Read an unsigned decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
x, X	Read a hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
e, f, g, E, F, G	Read a floating-point number, infinite value, or Not-a-Number (NaN) into the corresponding argument. The corresponding type is in accordance with the size character.
s	Read a string into a given array. The corresponding argument should be "char __far *arg*[ ]". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.

[ ]	Read a non-empty string into the memory area starting with argument *arg*. This area must be large enough to accommodate the string and the null character (\0) that is automatically appended to indicate the end of the string. The corresponding argument should be "char **arg*". The character pattern enclosed by [ ] can be used in place of the type specification character s. The character pattern is a character set that defines the search set of the characters constituting the input field of sscanf. If the first character within [ ] is "^", the search set is complemented, and all ASCII characters other than the characters within [ ] are included. In addition, a range specification feature that can be used as a shortcut is also available. For example, %[0-9] matches all decimal numbers. In this set, "-" cannot be specified as the first or last character. The character preceding "-" must be less in lexical sequence than the succeeding character. - %[abcd] Matches character strings that include only a, b, c, and d. - %[^abcd] Matches character strings that include any characters other than a, b, c, and d. - %[A-DW-Z] Matches character strings that include A, B, C, D, W, X, Y, and Z. - %[z-a] Matches z, -, and a (this is not considered a range specification).
c	Scan one character. The corresponding argument should be "char __far **arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
p	Store the pointer that was scanned. The corresponding argument should be "void __far ***arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Input data is not read. The number of characters that have been read so far is written to the corresponding parameter. Even though the %n directive is executed, the number of input items that are returned when the function ends is not increased. The corresponding type is in accordance with the size character.
%	Match the character "%". No conversion or assignment is performed. The conversion specification is "%%".

F conversion can only be specified for C99 libraries.

Make sure that a floating-point number (type specification characters e, f, g, E, F, and G) corresponds to thefollowing general format.

[+ | -] *ddddd* [.] *ddd* [E | e [+ | -] *ddd*]

However, the portions enclosed by [ ] in the above format are arbitrarily selected, and ddd indicates a decimal digit.

[Caution]

- scanf may stop scanning a specific field before the normal end-of-field character is reached or may stop completely.
- scanf stops scanning and storing a field and moves to the next field under the following conditions.
  - The substitution suppression character (*) appears after "%" in the format specification, and the input field at that point has been scanned but not stored.
  - A field width (positive decimal integer) specification character was read.
  - The character to be read next cannot be converted according to the conversion specification (for example, if Z is read when the specification is a decimal number).
  - The next character in the input field does not appear in the search set (or appears in the complement search set).

If scanf stops scanning the input field at that point because of any of the above reasons, it is assumed that the next character has not yet been read, and this character is used as the first character of the next field or the first character for the read operation to be executed after the input.

- scanf ends under the following conditions:
  - The next character in the input field does not match the corresponding ordinary character in the string to be converted.
  - The next character in the input field is EOF.
  - The string to be converted ends.
- If a list of characters that is not part of the conversion specification is included in the string to be converted, make sure that the same list of characters does not appear in the input. sscanf scans matching characters but does not store them. If there was a mismatch, the first character that does not match remains in the input as if it were not read.

[Restrictions]

a or A conversion and hexadecimal floating-point numbers are not supported.

vsnprintf [V1.07 or later]

Write text in specified format to array

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int __far vsnprintf(char __far * restrict *s*, size_t *n*, const char __far * restrict *format*, va_list *arg*); (C99)

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

From among the parameter sequence, this function writes the parameter indicated by pointer *arg* to the array indicated by *s*. The conversion method in this case complies with the format specified by the string indicated by *format*. When *n* is 0, no text is written and *s* may be a null pointer. In other cases, output characters subsequent to the (*n*-1)th character are discarded without being written to the array and the null character is written after the character string that was actually written to the array. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear.
For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99)
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99)
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99)
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99)
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries.

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99)
e, E	Convert a double type (float type in a single-precision function)argument to [-]d.ddddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99)
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99)
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.
p	Output the value of the pointer. The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

vsprintf

Write text in specified format to array

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far vsprintf(char __far *s, const char __far *format, va_list arg); (C90)
int __far vsprintf(char __far * restrict s, const char __far * restrict format, va_list arg); (C99) [V1.07 or later]
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Returns EOF(-1) if a write error has occurred.

[Description]

From among the parameter sequence, this function writes the parameter indicated by pointer *arg* to the string indicated by *s*. The conversion method in this case complies with the format specified by the string indicated by *format*. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives. The format will be output without change except for a conversion specification starting with %. The conversion specification fetches the 0 or more subsequent arguments, converts them, and then outputs them.

The *format* consists of the following two types of directives:

Ordinary characters	Characters that are copied directly without conversion (other than "%").
Conversion specifications	Specifications that fetch zero or more arguments and assign a specification.

Each conversion specification begins with character "%" (to insert "%" in the output, specify "%%" in the format string). The following appear after the "%":

%[flag][field-width][precision][size][type-specification-character]

The meaning of each conversion specification is explained below.

(1) flag
Zero or more flags, which qualify the meaning of the conversion specification, are placed in any order.
The flag characters and their meanings are as follows:

-	The result of the conversion will be left-justified in the field, with the right side filled with blanks (if this flag is not specified, the result of the conversion is right-justified).
+	The result of a signed conversion will start with a + or - sign (if this flag is not specified, the result of the conversion starts with a sign only when a negative value has been converted).
Space	If the first character of a signed conversion is not a sign and a signed conversion is not generated a character, a space (" ") will be appended to the beginning of result of the conversion. If both the space flag and + flag appear, the space flag is ignored.

#	The result is to be converted into an alternative format. For o conversion, the precision is increased so that the first digit of the conversion result is 0. For x or X conversion, 0x or 0X is appended to the beginning of a non-zero conversion result. For e, f, g, E, F, or G conversion, a decimal point "." is added to the conversion result even if no digits follow the decimal point[Note]. For g or G conversion, trailing zeros will not be removed from the conversion result. The operation is undefined for conversions other than the above.
0	For d, e, f, g, i, o, u, x, E, F, G, or X conversion, zeros are added following the specification of the sign or base to fill the field width. If both the 0 flag and - flag are specified, the 0 flag is ignored. For d, i, o, u, x, or X conversion, when the precision is specified, the zero (0) flag is ignored. Note that 0 is interpreted as a flag and not as the beginning of the field width. The operation is undefined for conversion other than the above.

Note Normally, a decimal point appears only when a digit follows it.

(2) field width

This is an optional minimum field width.
If the converted value is smaller than this field width, the left side is filled with spaces (if the left justification flag explained above is assigned, the right side will be filled with spaces). This field width takes the form of "*" or a decimal integer. If "*" is specified, an int type argument is used as the field width. A negative field width is not supported. If an attempt is made to specify a negative field width, it is interpreted as a minus (-) flag appended to the beginning of a positive field width.

(3) precision

For d, i, o, u, x, or X conversion, the value assigned for the precision is the minimum number of digits to appear. For e, f, E, or F conversion, it is the number of digits to appear after the decimal point. For g or G conversion, it is the maximum number of significant digits. For *s* conversion, it is the maximum number of bytes.
The precision takes the form of "*" or "." followed by a decimal integer. If "*" is specified, an int type argument is used as the precision. If a negative precision is specified, it is treated as if the precision were omitted. If only "." is specified, the precision is assumed to be 0. If the precision appears together with a conversion specification other than the above, the operation is undefined.

(4) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.
When hh is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a signed char or unsigned char argument. hh also causes a following n type specification to be forcibly applied to a pointer to a signed char argument. (C99) [V1.07 or later]
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short int or unsigned short int argument. h is also causes a following n type specification to be forcibly applied to a pointer to short argument.
When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long or unsigned long argument. l is also causes a following n type specification to be forcibly applied to a pointer to long argument.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.
When j is specified, a following d, i, o, u, x, or X type specification is forcibly applied to an intmax_t or uintmax_t argument. j also causes a following n type specification to be forcibly applied to a pointer to an intmax_t argument. (C99) [V1.07 or later]
When z is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a size_t or signed int argument. z also causes a following n type specification to be forcibly applied to a pointer to a signed int argument. (C99) [V1.07 or later]
When t is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a ptrdiff_t or unsigned int argument. t also causes a following n type specification to be forcibly applied to a pointer to a ptrdiff_t argument. (C99) [V1.07 or later]
When L is specified, a following e, E, f, F, g, or G type specification is forcibly applied to a long double argument. However, since the double type and long double type have the same format in this compiler, the specification has no effect.

(5) type specification character

These are characters that specify the type of conversion that is to be applied.
The characters that specify conversion types and their meanings are as follows.
F conversion can only be specified for C99 libraries. [V1.07 or later]

d, i	Convert an int type argument to a signed decimal number.

o, u, x, X	Convert an unsigned int type argument to octal notation (o), unsigned decimal notation (u), or unsigned hexadecimal notation (x or X) with dddd format. For x conversion, the letters abcdef are used. For X conversion, the letters ABCDEF are used.
f, F	Convert a double type (float type in a single-precision function) argument to decimal notation of the form [-]dddd.dddd. The format used for converting a double-type argument that indicates infinity is [-]inf for f conversion and [-]INF for F conversion. The format used for converting a double-type argument that indicates NaN is [-]nan for f conversion and [-]NAN for F conversion. (C99) [V1.07 or later]
e, E	Convert a double type (float type in a single-precision function)argument to [-]d.dddde±dd format, which has one digit before the decimal point (not 0 if the argument is not 0) and the number of digits after the decimal point is equal to the precision. The E conversion specification generates a number in which the exponent part starts with "E" instead of "e". The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
g, G	Convert a double type (float type in a single-precision function) argument to e (E for a G conversion specification) or f format, with the number of digits in the mantissa specified for the precision. Trailing zeros of the conversion result are excluded from the fractional part. The decimal point appears only when it is followed by a digit. The format for converting a double-type argument that indicates infinity or NaN is the same as the f conversion or F conversion specifier. (C99) [V1.07 or later]
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
s	The argument must be a pointer pointing to a character type array. Characters from this array are output up until the null character (\0) indicating termination (the null character (\0) itself is not included). If the precision is specified, no more than the specified number of characters will be output. If the precision is not specified or if the precision is greater than the size of this array, make sure that this array includes the null character (\0). The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.
p	Output the value of the pointer. The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
%	Output the character "%". No argument is converted. The conversion specification is "%%".

[Restrictions]

a conversion or A conversion of the C99 standard is not supported.

vsscanf [V1.08 or later]

Read text in specified format from character string

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int __far vsscanf(const char __far * restrict *s*, const char __far * restrict *format*, va_list *arg*); (C99) [V1.08 or later]

[Return value]

The number of input fields for which scanning, conversion, and storage were executed normally is returned. The return value does not include scanned fields that were not stored.
If an attempt is made to read to the end of the file, the return value is EOF.
If no field was stored, the return value is 0.

[Description]

This function converts the input from the string indicated by *s* and assigns the conversion result to the parameter indicated by pointer *arg* among the parameter sequence. The conversion method in this case complies with the format specified by the string indicated by *format*. When copying is executed between objects whose areas overlap, correct operation is not guaranteed.

Correct operation is not guaranteed if there is not enough arguments to satisfy the format. If the format becomes full even though arguments are still left, the extra arguments are merely evaluated and they will be ignored.
The format consists of 0 or more directives, and the directives in the format are executed in sequence. If there is no input character or execution of a directive fails due to an incorrect input, processing is terminated.

The *format* consists of the following three types of directives:

One or more Space characters	Space ( ), tab (\t), or new-line (\n). Reading of input data is executed up to immediately before the first non-white-space character (this character is left but not read) or until reading can no longer be performed.
Ordinary characters	All ASCII characters other than "%". Reading is executed by reading the next character.
Conversion specification	Fetches 0 or more arguments and directs the conversion.

Each conversion specification starts with "%". The following appear after the "%":

%[assignment-suppression-character][field-width][size][type-specification-character]

Each conversion specification is explained below.

(1) Assignment suppression character
The assignment suppression character "*" suppresses assignment of the input field.

(2) field width

This is a positive decimal integer that defines the maximum field width. When 0 is specified, there are no regulations.

It specifies the maximum number of characters that are read before the input field is converted. If the input field is smaller than this field width, sscanf reads all the characters in the field and then proceeds to the next field and its conversion specification.

If a space character or a character that cannot be converted is found before the number of characters equivalent to the field width is read, the characters up to the white space or the character that cannot be converted are read and stored. Then, sscanf proceeds to the next conversion specification.

(3) size

This is an arbitrary optional size character hh, h, l, ll, j, z, t, or L, which changes the default method for interpreting the data type of the corresponding argument.

If there is no specification, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to an int or unsigned int argument. Furthermore, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to an int pointer.

When hh is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a signed char or unsigned char argument. hh also causes a following f, F, e, E, g, or G type specification to be applied to a pointer to a float argument and a following n type specification to be applied to a pointer to a signed char argument. (C99) [V1.08 or later]

When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a short int or unsigned short int argument. When h is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a float argument, and an n type specification is forcibly applied to a short int pointer.

When l is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long or unsigned long argument. When l is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a double argument, and an n type specification is forcibly applied to a long pointer.

When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a pointer to a long long or unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer.

When j is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a intmax_t or uintmax_t argument, and an n type specification is forcibly applied to an intmax_t pointer. (C99) [V1.08 or later]

When z is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a size_t or signed int argument, and an n type specification is forcibly applied to an signed int pointer. (C99) [V1.08 or later]

When t is specified, a following d, i, o, n, u, x, or X type specification is forcibly applied to a pointer to a ptrdiff_t or unsigned int argument, and an n type specification is forcibly applied to an ptrdiff_t pointer. (C99) [V1.08 or later]

When L is specified, a following f, F, e, E, g, or G type specification is forcibly applied to a pointer to a long double argument. However, the double type and long double type have the same format in this compiler.

(4) type specification character

These are characters that specify the type of conversion that is to be applied.

The characters that specify conversion types and their meanings are as follows.

d	Read a decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
i	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
o	Read an octal integer into the corresponding argument. The corresponding type is in accordance with the size character.
u	Read an unsigned decimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
x, X	Read a hexadecimal integer into the corresponding argument. The corresponding type is in accordance with the size character.
e, f, g, E, F, G	Read a floating-point number, infinite value, or Not-a-Number (NaN) into the corresponding argument. The corresponding type is in accordance with the size character.
s	Read a string into a given array. The corresponding argument should be "char __far *arg*[ ]". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer. Correct operation is not guaranteed when a null pointer is passed.

[ ]	Read a non-empty string into the memory area starting with argument *arg*. This area must be large enough to accommodate the string and the null character (\0) that is automatically appended to indicate the end of the string. The corresponding argument should be "char **arg*". The character pattern enclosed by [ ] can be used in place of the type specification character s. The character pattern is a character set that defines the search set of the characters constituting the input field of sscanf. If the first character within [ ] is "^", the search set is complemented, and all ASCII characters other than the characters within [ ] are included. In addition, a range specification feature that can be used as a shortcut is also available. For example, %[0-9] matches all decimal numbers. In this set, "-" cannot be specified as the first or last character. The character preceding "-" must be less in lexical sequence than the succeeding character. - %[abcd] Matches character strings that include only a, b, c, and d. - %[^abcd] Matches character strings that include any characters other than a, b, c, and d. - %[A-DW-Z] Matches character strings that include A, B, C, D, W, X, Y, and Z. - %[z-a] Matches z, -, and a (this is not considered a range specification).
c	Scan one character. The corresponding argument should be "char __far **arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
p	Store the pointer that was scanned. The corresponding argument should be "void __far ***arg*". The pointer must always be the far pointer. When passing a constant, add a cast to the argument to clearly show that it is a pointer.
n	Input data is not read. The number of characters that have been read so far is written to the corresponding parameter. Even though the %n directive is executed, the number of input items that are returned when the function ends is not increased. The corresponding type is in accordance with the size character.
%	Match the character "%". No conversion or assignment is performed. The conversion specification is "%%".

F conversion can only be specified for C99 libraries.

Make sure that a floating-point number (type specification characters e, f, g, E, F, and G) corresponds to thefollowing general format.

[+ | -] *ddddd* [.] *ddd* [E | e [+ | -] *ddd*]

However, the portions enclosed by [ ] in the above format are arbitrarily selected, and ddd indicates a decimal digit.

[Caution]

- sscanf may stop scanning a specific field before the normal end-of-field character is reached or may stop completely.
- sscanf stops scanning and storing a field and moves to the next field under the following conditions.
  - The substitution suppression character (*) appears after "%" in the format specification, and the input field at that point has been scanned but not stored.
  - A field width (positive decimal integer) specification character was read.
  - The character to be read next cannot be converted according to the conversion specification (for example, if Z is read when the specification is a decimal number).
  - The next character in the input field does not appear in the search set (or appears in the complement search set).

If sscanf stops scanning the input field at that point because of any of the above reasons, it is assumed that the next character has not yet been read, and this character is used as the first character of the next field or the first character for the read operation to be executed after the input.

- sscanf ends under the following conditions:
  - The next character in the input field does not match the corresponding ordinary character in the string to be converted.
  - The next character in the input field is EOF.
  - The string to be converted ends.
- If a list of characters that is not part of the conversion specification is included in the string to be converted, make sure that the same list of characters does not appear in the input. sscanf scans matching characters but does not store them. If there was a mismatch, the first character that does not match remains in the input as if it were not read.

[Restrictions]

a or A conversion and hexadecimal floating-point numbers are not supported.

getchar

Read characters from SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far   getchar(void);
```

[Return value]

Returns the value read from SFR.

[Description]

This function reads a single character from P0 which is SFR. An error check is not performed for reading.

[Caution]

- To change stdin, replace this function.

gets

Read character string from SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
char    __near  *__far  gets(char __near *s);
char    __far   *__far  _COM_gets_f(char __far *s);
```

[Return value]

s is returned.

If the end of a file is detected but not a single character has been read into the array, the contents of the array are left without change and the null pointer is returned.

[Description]

This function reads a character string from SFR using the getchar function and stores the read data into the string indicated by *s*.

When the end of the file is detected or when a new-line character is read, reading of characters is terminated, the read new-line character is discarded, and finally a null character is written immediately after the character that was last stored in the array.

putchar

Write characters to SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int __far   putchar(int c);
```

[Return value]

The character *c* is returned.

[Description]

This function writes character *c* to P0 which is SFR. An error check is not performed for writing.

[Caution]

- To change stdout, replace this function. Note that replacing the putchar function will also change stderr. To change the output destination of stderr to something other than stdout, replace the perror function.

puts

Write character string to SFR

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int     __far  puts(const char __near *s);
int     __far   _COM_puts_f(const char __far *s);
```

[Return value]

0 is returned.
When the putchar function returns -1, this function returns -1.

[Description]

This function writes string *s* to SFR using the putchar function. The end-of-string null character is not write, but a new-line character is written in its place.

perror

Generate the error message

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
void    __far   perror(const char __near *s);
void    __far   p_COM_error_f(const char __far *s);
```

[Description]

This function outputs to stderr the error message that corresponds to global variable errno.
stderr which is the same as stdout becomes P0 which is SFR. It is output to SFR using the putchar function.
The message that is output is as follows.

When *s* is not NULL	`printf("%s:%s\n", s, s_fix);`
When *s* is NULL	`printf("%s\n", s_fix);`

s_fix is as follows.

When errno is 0	"No error"
When errno is EDOM	"EDOM error"
When errno is ERANGE	"ERANGE error"
Otherwise	"Unknown error"

[Caution]

- Note that replacing the putchar function will also change stderr. To change the output destination of stderr to something other than stdout, replace the perror function.

7.5.8 General utility functions

General utility functions are as follows.

Table 7.10 General Utility Functions

Function/Macro Name	Outline
atof	Conversion of character string to floating-point number (double type)
atoff	Conversion of character string to floating-point number (float type)
atoi	Conversion of character string to integer (int type)
atol	Conversion of character string to integer (long int type)
atoll [V1.07 or later]	Conversion of character string to integer (long long int type) (C99)
strtod	Conversion of character string to floating-point number (double type) (storing pointer to last character string
strtof	Conversion of character string to floating-point number (float type) (storing pointer to last character string
strtold [V1.07 or later]	Conversion of character string to floating-point number (long double type) (storing pointer to last character string) (C99)
strtol	Conversion of character string to integer (long int type) and storing pointer to last character string
strtoll [V1.07 or later]	Conversion of character string to integer (long long int type) and storing pointer to last character string (C99)
strtoul	Conversion of character string to integer (unsigned long int type) and storing pointer to last character string
strtoull [V1.07 or later]	Conversion of character string to integer (unsigned long long int type) and storing pointer to last character string (C99)
rand	Pseudorandom number sequence generation
srand	Setting of type of pseudorandom number sequence
calloc [V1.02 or later]	Allocates dynamic memory that is initialized by 0.
free [V1.02 or later]	Releases dynamic memory
malloc [V1.02 or later]	Allocates dynamic memory
realloc [V1.02 or later]	Re-allocates dynamic memory
abort	Terminates the program
bsearch	Binary search
qsort	Sort
abs	Output absolute value (int type)
div	Division (int type)
labs	Output absolute value (long type)
ldiv	Division (long type)
llabs [V1.07 or later]	Output absolute value (long long type) (C99)
lldiv [V1.07 or later]	Division (long long type) (C99)

atof

Conversion of character string to floating-point number (double type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
double  __far   atof(const char __near *nptr);
double  __far   _COM_atof_f(const char __far *nptr);
```

[Return value]

If the partial character string has been converted, the resultant value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, atof returns ±∞ and sets macro ERANGE to global variable errno.

If an underflow occurs, atof returns 0 and sets macro ERANGE to global variable errno.

[Description]

This function converts the first portion of the character string indicated by *nptr* into a float type representation.

atoff

Conversion of character string to floating-point number (float type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
float   __far   atoff(const char __near *nptr);
float   __far   _COM_atoff_f(const char __far *nptr);
```

[Return value]

If the partial character string has been converted, the resultant value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, atoff returns ±∞ and sets macro ERANGE to global variable errno.

If an underflow occurs, atoff returns 0 and sets macro ERANGE to global variable errno.

[Description]

This function converts the first portion of the character string indicated by *nptr* into a float type representation.

atoi

Conversion of character string to integer (int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
int __far   atoi(const char __near *nptr);
int __far   _COM_atoi_f(const char __far *nptr);
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, atoi returns INT_MAX for a positive value and INT_MIN for a negative value, and sets macro ERANGE to global variable errno.

[Description]

This function converts the first part of the character string indicated by nptr into an int type in the decimal representation.

atol

Conversion of character string to integer (long int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long int    __far   atol(const char __near *nptr);
long int    __far   _COM_atol_f(const char __far *nptr);
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, atol returns LONG_MAX for a positive value and LONG_MIN for a negative value, and sets macro ERANGE to global variable errno.

[Description]

This function converts the first part of the character string indicated by nptr into a long int type in the decimal representation.

atoll [V1.07 or later]

Conversion of character string to integer (long long int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long int __far atoll(const char __near *nptr); (C99)
long long int __far _COM_atoll_f(const char __far *nptr); (C99)
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, this function returns LLONG_MAX for a positive value and LLONG_MIN for a negative value, and sets macro ERANGE to global variable errno.

[Description]

This function converts the first part of the character string indicated by *nptr* into a long long int type in the decimal representation.

strtod

Conversion of character string to flonptrating-point number (double type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
double __far strtod(const char __near *nptr, char __near * __near *endptr); (C90)
double __far strtod(const char __near * restrict nptr, char __near * __near * restrict endptr); (C99) [V1.07 or later]
double __far _COM_strtod_ff(const char __far *nptr, char __far * __far *endptr); (C90)
double __far _COM_strtod_ff(const char __far * restrict nptr, char __far * __far * restrict endptr); (C99) [V1.07 or later]
```

[Return value]

If the partial character string has been converted, the resultant value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs , strtod returns ±∞ and sets macro ERANGE to global variable errno.

If an underflow occurs, strtod returns 0 and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a double-type representation. When *endptr* is not the null pointer, the pointer to the remaining strings that were not converted is set to *endptr*.

[Restrictions]

Hexadecimal floating-point numbers are not supported in a recognizable format of a string that is subject to the conversion method specified in the C99 standard.

strtof

Conversion of character string to flonptrating-point number (float type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
float __far strtof(const char __near *nptr, char __near * __near *endptr); (C90)
float __far strtof(const char __near * restrict nptr, char __near * __near * restrict endptr); (C99) [V1.07 or later]
float __far _COM_strtof_ff(const char __far *nptr, char __far * __far *endptr); (C90)
float __far _COM_strtof_ff(const char __far * restrict nptr, char __far * __far * restrict endptr); (C99) [V1.07 or later]
```

[Return value]

If the partial character string has been converted, the resultant value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, strtof returns ±∞ and sets macro ERANGE to global variable errno.

If an underflow occurs, strtof returns 0 and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a float-type representation. When *endptr* is not the null pointer, the pointer to the remaining strings that were not converted is set to *endptr*.

[Restrictions]

Hexadecimal floating-point numbers are not supported in a recognizable format of a string that is subject to the conversion method specified in the C99 standard.

strtold [V1.07 or later]

Conversion of character string to floating-point number (long double type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long double __far strtold(const char __near * restrict nptr, char __near * __near * restrict endptr); (C99)
long double __far _COM_strtold_ff(const char __far * restrict nptr, char __far * __far * restrict endptr); (C99)
```

[Return value]

If the partial character string has been converted, the converted value is returned. If the character string could not be converted, 0 is returned.

If an overflow occurs, this function returns ±∞ and sets macro ERANGE to global variable errno.

If an underflow occurs, this function returns 0 and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a long double-type representation. When *endptr* is not a null pointer, the pointer to the remaining strings that were not converted is set in *endptr*.

[Restrictions]

Hexadecimal floating-point numbers are not supported in a recognizable format of a string that is subject to the conversion method specified in the C99 standard.

strtol

Conversion of character string to integer (long int type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
long int __far strtol(const char __near *_nptr_, char __near * __near *_endptr_, int _base_); (C90)
long int __far strtol(const char __near * restrict _nptr_, char __near * __near * restrict _endptr_, int _base_); (C99) [V1.07 or later]
long int __far _COM_strtol_ff(const char __far *_nptr_, char __far * __far *_endptr_, int _base_); (C90)
long int __far _COM_strtol_ff(const char __far * restrict _nptr_, char __far * __far * restrict _endptr_, int _base_); (C99) [V1.07 or later]

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, strtol returns LONG_MAX or LONG_MIN and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a long int-type representation. If *base* is 0, the value is interpreted as the C radix representation. If *base* is between the range of 2 and 36, the value is interpreted as a radix. When *endptr* is not the null pointer, the pointer to the remaining strings that were not converted is set to *endptr*.

strtoll [V1.07 or later]

Conversion of character string to integer (long long int type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long int __far strtoll(const char __near * restrict nptr, char __near * __near * restrict endptr, int base); (C99)
long long int __far _COM_strtoll_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base); (C99)
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, this function returns LLONG_MAX or LLONG_MIN and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into a long long int-type representation. If *base* is 0, the value is interpreted as the C radix representation. If *base* is in the range from 2 to 36, the value is interpreted as a radix. When *endptr* is not a null pointer, the pointer to the remaining strings that were not converted is set in *endptr*.

strtoul

Conversion of character string to integer (unsigned long int type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
unsigned long int __far strtoul(const char __near *_nptr_, char __near * __near *_endptr_, int _base_); (C90)
unsigned long int __far strtoul(const char __near * restrict _nptr_, char __near * __near * restrict _endptr_, int _base_); (C99) [V1.07 or later]
unsigned long int __far _COM_strtoul_ff(const char __far *_nptr_, char __far * __far *_endptr_, int _base_); (C90)
unsigned long int __far _COM_strtoul_ff(const char __far * restrict _nptr_, char __far * __far * restrict _endptr_, int _base_); (C99) [V1.07 or later]

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, strtoul returns ULONG_MAX and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into an unsigned long int-type representation. If *base* is 0, the value is interpreted as the C radix representation. If *base* is between the range of 2 and 36, the value is interpreted as a radix. When *endptr* is not the null pointer, the pointer to the remaining strings that were not converted is set to *endptr*.

strtoull [V1.07 or later]

Conversion of character string to integer (unsigned long long int type) and storing pointer to last character string

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
unsigned long long int __far strtoull(const char __near * restrict *nptr*, char __near * __near * restrict *endptr*, int *base*); (C99)
unsigned long long int __far _COM_strtoull_ff(const char __far * restrict *nptr*, char __far * __far * restrict *endptr*, int *base*); (C99)

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.
If an overflow occurs, this function returns ULLONG_MAX and sets macro ERANGE to global variable errno.

[Description]

This function skips 0 or more columns of white-space characters (character which makes the isspace function true) from the start of the string indicated by *nptr*, and converts the string from the next character into an unsigned long long int-type representation. If *base* is 0, the value is interpreted as the C radix representation. If *base* is in the range from 2 to 36, the value is interpreted as a radix. When *endptr* is not a null pointer, the pointer to the remaining strings that were not converted is set in *endptr*.

rand

Pseudorandom number sequence generation

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
int __far   rand(void);
```

[Return value]

Random numbers are returned.

[Description]

This function returns a random number that is greater than or equal to zero and less than or equal to RAND_MAX (0x7FFF).

srand

Setting of type of pseudorandom number sequence

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __far  srand(unsigned int seed);
```

[Description]

This function assigns seed as the new pseudo random number sequence *seed* to be used by the rand call that follows. If srand is called using the same *seed* value, the same numbers in the same order will appear for the random numbers that are obtained by rand. If rand is executed without executing srand, the results will be the same as when srand(1) was first executed.

calloc [V1.02 or later]

Allocates memory that has been initialized by zero.

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __near * __far calloc(size_t nmemb, size_t size);
```

[Return value]

Upon succeeding to allocate an area, the pointer to that area is returned.
If *nmemb* or *size* is 0 or the area could not be allocated, a null pointer is returned.

[Description]

This function allocates an area whose size is specified by *size* and the number of elements in an array is specified by *nmemb* and then initializes that area by 0.

[Professional Edition only] [V1.03 or later]
When using a malloc library for the security facility, the __heap_chk_fail function is called when one of the following operations is performed.

- The pointer to an area other than that allocated by calloc, malloc, or realloc is passed to free or realloc.
- The pointer to an area released by free is passed again to free or realloc.
- After calloc, malloc, or realloc, a value is written to an address outside the allocated area (within two bytes before and after the allocated area) and the pointer to that area is passed to free or realloc.

The __heap_chk_fail function needs to be defined by the user and it describes the processing to be executed when an error occurs in management of dynamic memory.
Note the following points when defining the __heap_chk_fail function.

- The __heap_chk_fail function should be a far function whose return value and parameter type should be the void type.
  void __far __heap_chk_fail(void);
- Do not define the __heap_chk_fail function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.

The calloc, malloc, and realloc functions for the security facility allocate four extra bytes before and after each allocated area for the purpose of detecting writing to addresses outside the allocated area. This consumes more heap memory area than with the usual functions.

[Caution]

The default size of the heap memory area is 0x100 bytes.
To change the heap memory area, define the _REL_sysheap array and set the array size in the _REL_sizeof_sysheap variable.

```
[Example of setting the heap memory area]
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark The _REL_sysheap array should be allocated to an even address.

free [V1.02 or later]

Releases memory.

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __far free(void __near *ptr);
```

[Description]

This function releases the area indicated by *ptr*. If *ptr* is a null pointer, no processing is performed. In addition to that, if *ptr* is not an area allocated by calloc, malloc, or realloc, or *ptr* has already been released by free or realloc, operation is not guaranteed.

[Professional Edition only] [V1.03 or later]
When using a malloc library for the security facility, the __heap_chk_fail function is called when one of the following operations is performed.

- The pointer to an area other than that allocated by calloc, malloc, or realloc is passed to free or realloc.
- The pointer to an area released by free is passed again to free or realloc.
- After calloc, malloc, or realloc, a value is written to an address outside the allocated area (within two bytes before and after the allocated area) and the pointer to that area is passed to free or realloc.

The __heap_chk_fail function needs to be defined by the user and it describes the processing to be executed when an error occurs in management of dynamic memory.
Note the following points when defining the __heap_chk_fail function.

- The __heap_chk_fail function should be a far function whose return value and parameter type should be the void type.
  void __far __heap_chk_fail(void);
- Do not define the __heap_chk_fail function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.

The calloc, malloc, and realloc functions for the security facility allocate four extra bytes before and after each allocated area for the purpose of detecting writing to addresses outside the allocated area. This consumes more heap memory area than with the usual functions.

malloc [V1.02 or later]

Allocates memory.

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __near * __far malloc(size_t size);
```

[Return value]

Upon succeeding to allocate an area, the pointer to that area is returned.
If *size* is 0 or the area could not be allocated, a null pointer is returned.

[Description]

This function allocates an area whose size is specified by *size*. The area is not initialized.

[Professional Edition only] [V1.03 or later]
When using a malloc library for the security facility, the __heap_chk_fail function is called when one of the following operations is performed.

- The pointer to an area other than that allocated by calloc, malloc, or realloc is passed to free or realloc.
- The pointer to an area released by free is passed again to free or realloc.
- After calloc, malloc, or realloc, a value is written to an address outside the allocated area (within two bytes before and after the allocated area) and the pointer to that area is passed to free or realloc.

The __heap_chk_fail function needs to be defined by the user and it describes the processing to be executed when an error occurs in management of dynamic memory.
Note the following points when defining the __heap_chk_fail function.

- The __heap_chk_fail function should be a far function whose return value and parameter type should be the void type.
  void __far __heap_chk_fail(void);
- Do not define the __heap_chk_fail function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.

The calloc, malloc, and realloc functions for the security facility allocate four extra bytes before and after each allocated area for the purpose of detecting writing to addresses outside the allocated area. This consumes more heap memory area than with the usual functions.

[Caution]

The default size of the heap memory area is 0x100 bytes.
To change the heap memory area, define the _REL_sysheap array and set the array size in the _REL_sizeof_sysheap variable.

```
[Example of setting the heap memory area]
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark The _REL_sysheap array should be allocated to an even address.

realloc [V1.02 or later]

Re-allocates memory.

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __near * __far realloc(void __near *ptr, size_t size);
```

[Return value]

Upon succeeding to allocate an area, the pointer to that area is returned.
If *size* is 0 or the area could not be allocated, a null pointer is returned.

[Description]

This function changes the area indicated by *ptr* to the size specified by *size*.
The size before re-allocation and the contents of the area up to the smaller value of *size* do not change.
If the size is increased, the area for the increased part is not initialized.
If *ptr* is a null pointer, operation is the same as that for "malloc (*size*)".
If *ptr* is not a null pointer and *size* is 0, operation is the same as that for "free (*ptr*)".
In addition to that, if *ptr* is not an area allocated by calloc, malloc, or realloc, or *ptr* has already been released by free or realloc, operation is not guaranteed.

[Professional Edition only] [V1.03 or later]
When using a malloc library for the security facility, the __heap_chk_fail function is called when one of the following operations is performed.

- The pointer to an area other than that allocated by calloc, malloc, or realloc is passed to free or realloc.
- The pointer to an area released by free is passed again to free or realloc.
- After calloc, malloc, or realloc, a value is written to an address outside the allocated area (within two bytes before and after the allocated area) and the pointer to that area is passed to free or realloc.

The __heap_chk_fail function needs to be defined by the user and it describes the processing to be executed when an error occurs in management of dynamic memory.
Note the following points when defining the __heap_chk_fail function.

- The __heap_chk_fail function should be a far function whose return value and parameter type should be the void type.
  void __far __heap_chk_fail(void);
- Do not define the __heap_chk_fail function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.

The calloc, malloc, and realloc functions for the security facility allocate four extra bytes before and after each allocated area for the purpose of detecting writing to addresses outside the allocated area. This consumes more heap memory area than with the usual functions.

[Caution]

The default size of the heap memory area is 0x100 bytes.
To change the heap memory area, define the _REL_sysheap array and set the array size in the _REL_sizeof_sysheap variable.

```
[Example of setting the heap memory area]
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark The _REL_sysheap array should be allocated to an even address.

abort

Terminates the program

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __fat  abort(void);
```

[Description]

Calling abort terminates the program. Control does not return to the calling function.

bsearch

Search for a value from among arranged arrays.

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
void __near * __far bsearch(const void __near **key*, const void __near **base*, size_t *nmemb,* size_t *size*, int (__far **compar*)(const void __near *, const void __near *));
void __far * __far _COM_bsearch_f(const void __far **key*, const void __far **base*, size_t *nmemb,*, size_t *size*, int (__far **compar*)(const void __far *, const void __far *));

[Return value]

A pointer to the element in the array that coincides with *key* is returned. If there are two or more elements that coincide with *key*, the one that has been found first is indicated. If there are not elements that coincide with *key*, a null pointer is returned.

[Description]

This function searches an element that coincides with *key* from an array starting with *base* by means of binary search. *nmemb* is the number of elements of the array. *size* is the size of each element.
The compare function indicated by *compar* is called with the pointer to object *key* as the first parameter and the pointer to array elements as the second parameter. As a result, if the first parameter is smaller than the second parameter, a negative value is returned. If the two parameters are equal, 0 is returned. If the first parameter is greater than the second parameter, a positive value is returned. The array must be arranged in the ascending order in respect to the compare function indicated by *compar* (last argument).

qsort

Sorts the array

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void __far  qsort(void __near *base, size_t nmemb, size_t size, int (__far *compar)(const void __near *, const void __near *));
void __far  _COM_qsort_f(void __far *base, size_t nmemb, size_t size, int (__far *compar)(const void __far *, const void __far *));
```

[Description]

This function sorts the array pointed to by *base* into ascending order in relation to the comparison function pointed to by *compar*. *nmemb* is the number of array elements, and *size* is the size of each element. The comparison function pointed to by *compar* is the same as the one described for bsearch.

If two elements are equal, their order when they are aligned in the array cannot be guaranteed.

abs

Obtain absolute value (int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
int __far   abs(int j);
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |. If the input value of abs is the smallest negative value, the same value is returned.

[Description]

This function obtains the absolute value of *j* (size of *j*), | *j* |. If *j* is a negative number, the result is the reversal of j. If *j* is not negative, the result is *j*.

div

Perform division of int type to obtain the quotient and remainder

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
div_t __far div(int numer, int denom);
```

[Return value]

The structure holding the result of the division is returned. When divided by 0, -1 is set as quotient quot and *numer* is set as remainder rem.

[Description]

This function is used to divide a value of int type.

This function calculates the quotient (quot) and remainder (rem) resulting from dividing numerator *numer* by denominator *denom*, and stores these two integers as the members of the following structure div_t.

```
typedef struct {
        int quot;
        int rem;
} div_t;
```

When the value cannot be divided, the quotient of the result becomes an integer that is closest to the algebraical quotient and has a smaller absolute value than it.

labs

Obtain absolute value (long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long int __far  labs(long int j);
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |. If the input value of labs is the smallest negative value, the same value is returned.

[Description]

This function obtains the absolute value of *j* (size of *j*), | *j* |. If *j* is a negative number, the result is the reversal of j. If *j* is not negative, the result is *j*. This function is the same as abs, but uses long type instead of int type, and the return value is also of long type.

ldiv

Perform division of long type to obtain the quotient and remainder

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
ldiv_t ldiv(long *numer*, long *denom*);

[Return value]

The structure holding the result of the division is returned. When divided by 0, -1 is set as quotient quot and *numer* is set as remainder rem.

[Description]

This function is used to divide a value of long type.
This function calculates the quotient (quot) and remainder (rem) resulting from dividing numerator *numer* by denominator *denom*, and stores these two integers as the members of the following structure div_t.

```
typedef struct {
        long int    quot;
        long int    rem;
} ldiv_t;
```

When the value cannot be divided, the quotient of the result becomes an integer that is closest to the algebraical quotient and has a smaller absolute value than it.

llabs [V1.07 or later]

Obtain absolute value (long long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long int __far llabs(long long int j); (C99)
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |. If the input value of llabs is the smallest negative value, the same value is returned.

[Description]

This function obtains the absolute value of *j* (size of *j*), | *j* |. If *j* is a negative number, the result is the reversal of *j*. If *j* is not negative, the result is *j*.

lldiv [V1.07 or later]

Perform division of long long type to obtain the quotient and remainder

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
lldiv_t __far lldiv(long long int *numer*, long long int *denom*); (C99)

[Return value]

The structure holding the result of the division is returned. When divided by 0, -1 is set as quotient quot and *numer* is set as remainder rem.

[Description]

This function is used to divide a value of long long type.
This function calculates the quotient (quot) and remainder (rem) resulting from dividing numerator *numer* by denominator *denom*, and stores these two integers as the members of the following structure lldiv_t.

```
typedef struct {
    long long int quot;
    long long int rem;
} lldiv_t;
```

When the value cannot be divided, the quotient of the result becomes an integer that is closest to the algebraical quotient and has a smaller absolute value than it.

7.5.9 Character string operation functions

Character string operation functions are as follows.

Table 7.11 Character String Operation Functions

Function/Macro Name	Outline
memcpy	Memory copy
memmove	Memory move
strcpy	Character string copy
strncpy	Character string copy (with number of characters specified)
strcat	Character string concatenation
strncat	Character string concatenation (with number of characters specified)
memcmp	Memory comparison
strcmp	Character string comparison
strncmp	Character string comparison (with number of characters specified)
memchr	Memory search
strchr	Character string search (start position of specified character)
strcspn	Character string search (maximum length not including specified character)
strpbrk	Character string search (start position)
strrchr	Character string search (end position)
strspn	Character string search (maximum length including specified character)
strstr	Character string search (start position of specified character string)
strtok	Token division
memset	Initialization of an object
strerror	Obtain error message corresponding to error number
strlen	Length of character string

memcpy

Object copy

[Classification]

Standard library

[Syntax]

#include <string.h>
void __near * __far memcpy(void __near *s1, const void __near *s2, size_t n); (C90)
void __near * __far memcpy(void __near * restrict s1, const void __near * restrict s2, size_t n); (C99) [V1.07 or later]
void __far * __far _COM_memcpy_ff(void __far *s1, const void __far *s2, size_t n); (C90)
void __far * __far _COM_memcpy_ff(void __far * restrict s1, const void __far * restrict s2, size_t n); (C99) [V1.07 or later]

[Return value]

Returns the value of *s1*.

[Description]

This function copies the *n* number of characters from a object indicated by *s2* to a object indicated by *s1*.
The operation is undefined if the copy source and copy destination areas overlap.

[Caution]

The memcpy function may be internally called by CC-RL and is included in the runtime library. Do not create a user function having the name memcpy.

memmove

Object copy

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void __near * __far memmove(void __near *s1, void __near *s2, size_t n);
void __far * __far  _COM_memmove_ff(void __far *s1, void __far *s2, size_t n);
```

[Return value]

Returns the value of *s1*.

[Description]

This function copies the *n* number of characters from a memory area indicated by *s2* to a memory area indicated by *s1*. Copying can be performed correctly even when the copy source area overlaps with the copy destination area.

strcpy

Character string copy

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strcpy(char __near *s1, const char __near *s2); (C90)
char __near * __far strcpy(char __near * restrict s1, const char __near * restrict s2); (C99) [V1.07 or later]
char __far * __far _COM_strcpy_ff(char __far *s1, const char __far *s2); (C90)
char __far * __far _COM_strcpy_ff(char __far * restrict s1, const char __far * restrict s2); (C99) [V1.07 or later]
```

[Return value]

Returns the value of *s1*.

[Description]

This function copies the character string indicated by *s2* (terminating null character is included) to the array indicated by *s1*.

The operation is undefined if the copy source and copy destination areas overlap.

strncpy

Character string copy with number of characters specified

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strncpy(char __near *s1, const char __near *s2, size_t n); (C90)
char __near * __far strncpy(char __near * restrict s1, const char __near * restrict s2, size_t n); (C99) [V1.07 or later]
char __far * __far _COM_strncpy_ff(char __far *s1, const char __far *s2, size_t n); (C90)
char __far * __far _COM_strncpy_ff(char __far * restrict s1, const char __far * restrict s2, size_t n); (C99) [V1.07 or later]
```

[Return value]

Returns the value of *s1*.

[Description]

This function copies up to *n* characters (null character and string following the null character are not appended) from the array indicated by *s2* to the array indicated by *s1*. If the array indicate by *s2* is shorter than *n* characters, null characters (\0) are appended to the duplication in the array indicated by *s1*, until all *n* characters are written. If the array indicated by *s2* is equal to or greater than *n* characters, null characters are not appended.

The operation is undefined if the copy source and copy destination areas overlap.

strcat

Character string concatenation

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strcat(char __near *s1, const char __near *s2); (C90)
char __near * __far strcat(char __near * restrict s1, const char __near * restrict s2); (C99) [V1.07 or later]
char __far * __far _COM_strcat_ff(char __far *s1, const char __far *s2); (C90)
char __far * __far _COM_strcat_ff(char __far * restrict s1, const char __far * restrict s2); (C99) [V1.07 or later]
```

[Return value]

Returns the value of *s1*.

[Description]

This function concatenates the duplication of the character string indicated by *s2* to the end of the character string indicated by *s1*, including the null character (\0). The first character of *s2* overwrites the null character (\0) at the end of *s1*.

The operation is undefined if the copy source and copy destination areas overlap.

strncat

Character string concatenation with number of characters specified

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strncat(char __near *s1, const char __near *s2, size_t n); (C90)
char __near * __far strncat(char __near * restrict s1, const char __near * restrict s2, size_t n); (C99) [V1.07 or later]
char __far * __far _COM_strncat_ff(char __far *s1, const char __far *s2, size_t n); (C90)
char __far * __far _COM_strncat_ff(char __far * restrict s1, const char __far * restrict s2, size_t n); (C99) [V1.07 or later]
```

[Return value]

Returns the value of *s1*.

[Description]

This function concatenates up to *n* characters (null character and string following the null character are not appended) to the end of the character string indicated by *s1*, starting from the beginning of the array indicated by *s2*. The null character (\0) at the end of *s1* is written over the first character of *s2*. The null character indicating termination (\0) is always added to this result.

The operation is undefined if the copy source and copy destination areas overlap.

[Caution]

Because the null character (\0) is always appended when strncat is used, if copying is limited by the number of *n* arguments, the number of characters appended to *s1* is *n* + 1.

memcmp

Object comparison

[Classification]

Standard library

[Syntax]

```
#include <string.h>
int __far   memcmp(const void __near *s1, const void __near *s2, size_t n);
int __far   _COM_memcmp_ff(const void __far *s1, const void __far *s2, size_t n);
```

[Return value]

An value greater than, equal to, or less than 0 is returned, depending on whether the object indicated by *s1* is greater than, equal to, or less than the object indicated by *s2*.

[Description]

This function compares the first n characters of an object indicated by *s1* with the object indicated by *s2*.

strcmp

Character string comparison

[Classification]

Standard library

[Syntax]

```
#include <string.h>
int __far   strcmp(const char __near *s1, const char __near *s2);
int __far   _COM_strcmp_ff(const char __far *s1, const char __far *s2);
```

[Return value]

Returns an value greater than, equal to, or less than 0, depending on whether the character string indicated by *s1* is greater than, equal to, or less than the character string indicated by *s2*.

[Description]

This function compares the character string indicated by *s1* with the character string indicated by *s2*.

strncmp

Character string comparison with number of characters specified

[Classification]

Standard library

[Syntax]

```
#include <string.h>
int __far   strncmp(const char __near *s1, const char __near *s2, size_t n);
int __far   _COM_strncmp_ff(const char __far *s1, const char __far *s2, size_t n);
```

[Return value]

Returns an value greater than, equal to, or less than 0, depending on whether the array indicated by *s1* is greater than, equal to, or less than the array indicated by *s2*.

[Description]

This function compares up to *n* characters of the array indicated by *s1* with characters of the array indicated by *s2* (string following the null character is not compared).

memchr

Character search from object

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void __near * __far memchr(const void __near *s, int c, size_t n);
void __far * __far  _COM_memchr_f(const void __far *s, int c, size_t n);
```

[Return value]

If *c* is found, a pointer indicating this character is returned. If *c* is not found, the null pointer is returned.

[Description]

This function obtains the position at which character *c* appears first in the first *n* number of characters in an object indicated by *s*.

strchr

Search for a character from the start of the string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strchr(const char __near *s, int c);
char __far * __far  _COM_strchr_f(const char __far *s, int c);
```

[Return value]

Returns a pointer indicating the character that has been found. If *c* does not appear in this character string, the null pointer is returned.

[Description]

This function obtains the position at which a character the same as *c* appears in the character string indicated by *s*. The null character (\0) indicating termination is regarded as part of this character string.

strcspn

Obtain the length of the start portion that does not include the specified character string among the strings

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t __fat    strcspn(const char __near *s1, const char __near *s2);
size_t __fat    _COM_strcspn_ff(const char __far *s1, const char __far *s2);
```

[Return value]

Returns the length of the portion that has been found.

[Description]

This function obtains the length of the maximum and first portion consisting of characters missing from the character string indicated by *s2* (except the null character (\0) at the end) in the character string indicated by *s1*.

strpbrk

Search for any specified character in the string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strpbrk(const char __near *s1, const char __near *s2);
char __far * __far  _COM_strpbrk_ff(const char __far *s1, const char __far *s2);
```

[Return value]

Returns the pointer to the character searched for. If any of the characters from *s2* does not appear in *s1*, the null pointer is returned.

[Description]

This function obtains the position in the character string indicated by *s1* at which any of the characters in the character string indicated by *s2* (except the null character (\0)) appears first.

strrchr

Character search from end position of character string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strrchr(const char __near *s, int c);
char __far * __far  _COM_strrchr_f(const char __far *s, int c);
```

[Return value]

Returns a pointer indicating character that has been found. If *c* does not appear in this character string, the null pointer is returned.

[Description]

This function obtains the position at which *c* appears last in the character string indicated by *s*. The null character (\0) indicating termination is regarded as part of this character string.

strspn

Obtain the length of the start portion that includes the specified character string among the strings

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t __far    strspn(const char __near *s1, const char __near *s2);
size_t __far    _COM_strspn_ff(const char __far *s1, const char __far *s2);
```

[Return value]

Returns the length of the portion that has been found.

[Description]

This function obtains the maximum and first length of the portion consisting of only the characters (except the null character (\0)) in the character string indicated by *s2*, in the character string indicated by *s1*.

strstr

Character string search from character string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __near * __far strstr(const char __near *s1, const char __near *s2);
char __far * __far  _COM_strstr_ff(const char __far *s1, const char __far *s2);
```

[Return value]

Returns the pointer indicating the character string that has been found. If character string indicated by *s2* is not found, the null pointer is returned. If *s2* indicates a character string with a length of 0, *s1* is returned.

[Description]

This function obtains the position of the portion (except the null character (\0)) that first coincides with the character string indicated by *s2*, in the character string indicated by *s1*.

strtok

Token division

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __far * __far strtok(char __far *s1, const char __far *s2); (C90)
char __far * __far strtok(char __far * restrict s1, const char __far * restrict s2); (C99) [V1.07 or later]
```

[Return value]

Returns a pointer to a token. If a token does not exist, the null pointer is returned.

[Description]

This function divides the character string indicated by *s1* into strings of tokens by delimiting the character string with a character in the character string indicated by *s2*. If this function is called first, *s1* is used as the first argument. Then, calling with the null pointer as the first argument continues. The delimiting character string indicated by *s2* can differ on each call.

On the first call, the character string indicated by s is searched for the first character not included in the delimiting character string indicated by *s2*. If such a character is not found, a token does not exist in the character string indicated by *s1*, and strtok returns the null pointer. If a character is found, that character is the beginning of the first token.

After that, strtok searches from the position of that character for a character included in the delimiting character string at that time. If such a character is not found, the token is expanded to the end of the character string indicated by *s1*, and the subsequent search returns the null pointer. If a character is found, the subsequent character is overwritten by the null character (\0) indicating the termination of the token. strtok saves the pointer indicating the subsequent character. The next search for a token starts from there.

In the second or subsequent call with the null pointer as the first argument, the search starts from where the retained pointer indicates. If the null pointer is used as the value of the first argument, a code that is not reentrancy is returned.

memset

Initialization with the specified character from the start of an object for the specified number of characters

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void __near * __far memset(const void __near *s, int c, size_t n);
void __far * __far  _COM_memset_f(const void __far *s, int c, size_t n);
```

[Return value]

Returns the value of *s*.

[Description]

This function copies the value of *c* to the first *n* character of an object indicated by *s*.

[Caution]

The memset function may be internally called by CC-RL and is included in the runtime library. Do not create a user function having the name memset.

strerror

Obtain error message corresponding to error number

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char __far * __far  strerror(int errnum);
```

[Return value]

Returns the pointer to the string of the error message corresponding to *errnum*.
If there is no corresponding error message, the null pointer is returned.

[Description]

Obtain error message corresponding to error number *errnum*. The error message should not be changed from the application program.

strlen

Length of character string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t __far    strlen(const char __near *s);
size_t __far    _COM_strlen_f(const char __far *s);
```

[Return value]

Returns the number of characters existing before the null character (\0) indicating termination.

[Description]

This function obtains the length of the character string indicated by *s*.

7.5.10 Initialization functions

Initialization functions are as follows.

Table 7.12 Initialization Function

Function/Macro Name	Outline
hdwinit	Initialization of peripheral devices immediately after the CPU reset
stkinit	Initialization of stack area

hdwinit

Initialization of peripheral devices immediately after the CPU reset.

[Classification]

Other library

[Syntax]

```
#include <_h_c_lib.h>
void __far      hdwinit(void);
```

[Description]

This function performs initialization of peripheral devices immediately after the CPU reset.
This is called from inside the startup routine.
The function included in the library is a dummy routine that performs no actions; code a function in accordance with your system.

stkinit

Initialization of stack area

[Classification]

Other library

[Syntax]

```
#include <_h_c_lib.h>
void __far     stkinit(void __near * stackbss);
```

[Description]

This function performs initialization of stack area.
This is called from inside the startup routine.
The lower 16 bits of the start address for the stack area are passed to the parameter.
When parity error detection by reading uninitialized RAM is not performed, this function does not have to be called.

7.5.11 Runtime libraries

Runtime libraries are as follows.

Table 7.13 Runtime Libraries

Function Name	Outline
_COM_fadd	float type addition
_COM_dadd	double type (double precision) addition
_COM_fsub	float type subtraction
_COM_dsub	double type (double precision) subtraction
_COM_imul	int type multiplication
_COM_lmul	long type multiplication
_COM_llmul	long long type multiplication
_COM_fmul	float type multiplication
_COM_dmul	double type (double precision) multiplication
_COM_mulsi	signed int type multiplication (result is signed long type)
_COM_mului	unsigned int type multiplication (result is unsigned long type)
_COM_mulsl	signed long type multiplication (result is signed long long type)
_COM_mulul	unsigned long type multiplication (result is unsigned long long type)
_COM_scdiv	signed char type division
_COM_ucdiv	unsigned char type division
_COM_sidiv	signed int type division
_COM_uidiv	unsigned int type division
_COM_sldiv	signed long type division
_COM_uldiv	unsigned long type division
_COM_slldiv	signed long long type division
_COM_ulldiv	unsigned long long type division
_COM_fdiv	float type division
_COM_ddiv	double type (double precision) division
_COM_divui	unsigned int type division (divisor is unsigned char type)
_COM_divul	unsigned long type division (divisor is unsigned int type)
_COM_screm	signed char type remainder operation
_COM_ucrem	unsigned char type remainder operation
_COM_sirem	signed int type remainder operation
_COM_uirem	unsigned int type remainder operation
_COM_slrem	signed long type remainder operation
_COM_ulrem	unsigned long type remainder operation
_COM_sllrem	signed long long type remainder operation

Function Name	Outline
_COM_ullrem	unsigned long long type remainder operation
_COM_remui	unsigned int type remainder operation (divisor is unsigned char type)
_COM_remul	unsigned long type remainder operation (divisor is unsigned int type)
_COM_macsi	signed int type multiply-accumulate operation (operation result is signed long type)
_COM_macui	unsigned int type multiply-accumulate operation (operation result is unsigned long type)
_COM_lshl	long type left-shift
_COM_llshl	long long type left-shift
_COM_lshr	long type logical right-shift
_COM_llshr	long long type logical right-shift
_COM_lsar	long type arithmetic right-shift
_COM_llsar	long long type arithmetic right-shift
_COM_feq	float type comparison (==)
_COM_deq	double type (double precision) comparison (==)
_COM_fne	float type comparison (!=)
_COM_dne	double type (double precision) comparison (!=)
_COM_fge	float type comparison (>=)
_COM_dge	double type (double precision) comparison (>=)
_COM_flt	float type comparison (<)
_COM_dlt	double type (double precision) comparison (<)
_COM_fle	float type comparison (<=)
_COM_dle	double type (double precision) comparison (<=)
_COM_fgt	float type comparison (>)
_COM_dgt	double type (double precision) comparison (>)
_COM_funordered	float type NaN test
_COM_dunordered	double type (double precision) NaN test
_COM_sltof	Type conversion from signed long type to float type
_COM_sltod	Type conversion from signed long type to double type (double precision)
_COM_ultof	Type conversion from unsigned long type to float type
_COM_ultod	Type conversion from unsigned long type to double type (double precision)
_COM_slltof	Type conversion from signed long long type to float type
_COM_slltod	Type conversion from signed long long type to double type (double precision)
_COM_ulltof	Type conversion from unsigned long long type to float type
_COM_ulltod	Type conversion from unsigned long long type to double type (double precision)
_COM_ftosl	Type conversion from float type to signed long type
_COM_ftoul	Type conversion from float type to unsigned long type

Function Name	Outline
_COM_ftosll	Type conversion from float type to signed long long type
_COM_ftoull	Type conversion from float type to unsigned long long type
_COM_dtosl	Type conversion from double type (double precision) to signed long type
_COM_dtoul	Type conversion from double type (double precision) to unsigned long type
_COM_dtosll	Type conversion from double type (double precision) to signed long long type
_COM_dtoull	Type conversion from double type (double precision) to unsigned long long type
_COM_ftod	Type conversion from float type to double type (double precision)
_COM_dtof	Type conversion from double type (double precision) to float type
__control_flow_integrity [Professional Edition only] [V1.06 or later]	Checks for indirect function calls.

7.6 Interrupt Disabled Time, Use of Data Sections, and Reentrancy

This section explains the interrupt disabled time and reentrancy of each function in the library.

7.6.1 Standard library

The interrupt disabled time, use of the initialized data section (.data), use of the uninitialized data section (.bss), and the reentrancy of each function in the standard library are shown in the following.

The interrupt disabled time is displayed, starting from the left, for the case of not using division/multiplication and multiply-accumulate units, the case of using division/multiplication and multiply-accumulate units, the case of using division/multiplication extended instructions (single precision), and the case of using division/multiplication extended instructions (double precision). A single numerical value is common for all libraries.

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reentrancy	Remark (non-reentrancy source)
assert	0	X	X	O	
isalnum	0	X	X	O	
isalpha	0	X	X	O	
isascii	0	X	X	O	
isblank [V1.07 or later]	0	X	X	O	(C99)
iscntrl	0	X	X	O	
isdigit	0	X	X	O	
isgraph	0	X	X	O	
islower	0	X	X	O	
isprint	0	X	X	O	
ispunct	0	X	X	O	
isspace	0	X	X	O	
isupper	0	X	X	O	
isxdigit	0	X	X	O	

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
toascii	0	X	X	O	
tolower	0	X	X	O	
toupper	0	X	X	O	
imaxabs [V1.07 or later]	0	X	X	O	(C99)
imaxdiv [V1.07 or later]	0/42/0/0	X	X	O	(C99)
strtoimax [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
_COM_strtoimax_ff [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
strtoumax [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
_COM_strtoumax_ff [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
fpclassify [V1.08 or later]	0	X	X	O	(C99)
isfinite [V1.08 or later]	0	X	X	O	(C99)
isinf [V1.08 or later]	0	X	X	O	(C99)
isnan [V1.08 or later]	0	X	X	O	(C99)
isnormal [V1.08 or later]	0	X	X	O	(C99)
signbit [V1.08 or later]	0	X	X	O	(C99)
acos	0/41/0/0	O	X	X	errno
acosf	0/41/0/0	O	X	X	errno
acosl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
asin	0/41/0/0	O	X	X	errno
asinf	0/41/0/0	O	X	X	errno
asinl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
atan	0/41/0/0	O	X	X	errno
atanf	0/41/0/0	O	X	X	errno
atanl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
atan2	0/41/0/0	O	X	X	errno
atan2f	0/41/0/0	O	X	X	errno
atan2l [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
cos	0/14/0/0	O	X	X	errno
cosf	0/14/0/0	O	X	X	errno
cosl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
sin	0/14/0/0	O	X	X	errno
sinf	0/14/0/0	O	X	X	errno
sinl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
tan	0/41/0/0	O	X	X	errno
tanf	0/41/0/0	O	X	X	errno
tanl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
acosh [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
acoshf [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
acoshl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
asinh [V1.08 or later]	0/41/0/0	O	X	O	(C99)
asinhf [V1.08 or later]	0/41/0/0	O	X	O	(C99)
asinhl [V1.08 or later]	0/41/0/0	O	X	O	(C99)
atanh [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
atanhf [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
atanhl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
cosh	0/41/0/0	O	X	X	errno
coshf	0/41/0/0	O	X	X	errno
coshl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
sinh	0/41/0/0	O	X	X	errno
sinhf	0/41/0/0	O	X	X	errno
sinhl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
tanh	0/41/0/0	O	X	X	errno
tanhf	0/41/0/0	O	X	X	errno
tanhl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
exp	0/41/0/0	O	X	X	errno
expf	0/41/0/0	O	X	X	errno
expl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
frexp	0/14/0/0	O	X	X	errno
frexpf	0/14/0/0	O	X	X	errno
frexpl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
ldexp	0/14/0/0	O	X	X	errno
ldexpf	0/14/0/0	O	X	X	errno
ldexpl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
log	0/14/0/0	O	X	X	errno
logf	0/14/0/0	O	X	X	errno
logl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
log10	0/14/0/0	O	X	X	errno

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
log10f	0/14/0/0	O	X	X	errno
log10l [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
log1p [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
log1pf [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
log1pl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
modf	0	O	X	X	errno
modff	0	O	X	X	errno
modfl [V1.08 or later]	0	O	X	X	errno (C99)
scalbn [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
scalbnf [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
scalbnl [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
scalbln [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
scalblnf [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
scalblnl [V1.09 or later]	0/14/0/0	O	X	X	errno (C99)
fabs	0	O	X	X	errno
fabsf	0	O	X	X	errno
fabsl [V1.08 or later]	0	O	X	X	errno (C99)
pow	0/41/0/0	O	X	X	errno
powf	0/41/0/0	O	X	X	errno
powl [V1.08 or later]	0/41/0/0	O	X	X	errno (C99)
sqrt	0/24/0/0	O	X	X	errno
sqrtf	0/24/0/0	O	X	X	errno
sqrtl [V1.08 or later]	0/24/0/0	O	X	X	errno (C99)
ceil	0	O	X	X	errno
ceilf	0	O	X	X	errno
ceill [V1.08 or later]	0	O	X	X	errno (C99)
floor	0	O	X	X	errno
floorf	0	O	X	X	errno
floorl [V1.08 or later]	0	O	X	X	errno (C99)
nearbyint [V1.09 or later]	0	X	X	O	(C99)
nearbyintf [V1.09 or later]	0	X	X	O	(C99)
nearbyintl [V1.09 or later]	0	X	X	O	(C99)
rint [V1.09 or later]	0	X	X	O	(C99)
rintf [V1.09 or later]	0	X	X	O	(C99)

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
rintl [V1.09 or later]	0	X	X	O	(C99)
lrint [V1.09 or later]	0	X	X	X	errno (C99)
lrintf [V1.09 or later]	0	X	X	X	errno (C99)
lrintl [V1.09 or later]	0	X	X	X	errno (C99)
llrint [V1.09 or later]	0	X	X	X	errno (C99)
llrintf [V1.09 or later]	0	X	X	X	errno (C99)
llrintl [V1.09 or later]	0	X	X	X	errno (C99)
round [V1.09 or later]	0	X	X	O	(C99)
roundf [V1.09 or later]	0	X	X	O	(C99)
roundl [V1.09 or later]	0	X	X	O	(C99)
lround [V1.09 or later]	0	X	X	X	errno (C99)
lroundf [V1.09 or later]	0	X	X	X	errno (C99)
lroundl [V1.09 or later]	0	X	X	X	errno (C99)
llround [V1.09 or later]	0	X	X	X	errno (C99)
llroundf [V1.09 or later]	0	X	X	X	errno (C99)
llroundl [V1.09 or later]	0	X	X	X	errno (C99)
trunc [V1.09 or later]	0	X	X	O	(C99)
truncf [V1.09 or later]	0	X	X	O	(C99)
truncl [V1.09 or later]	0	X	X	O	(C99)
fmod	0/14/0/0	O	X	X	errno
fmodf	0/14/0/0	O	X	X	errno
fmodl [V1.08 or later]	0/14/0/0	O	X	X	errno (C99)
copysign [V1.09 or later]	0	X	X	O	(C99)
copysignf [V1.09 or later]	0	X	X	O	(C99)
copysignl [V1.09 or later]	0	X	X	O	(C99)
nan [V1.09 or later]	0	X	X	O	(C99)
nanf [V1.09 or later]	0	X	X	O	(C99)
nanl [V1.09 or later]	0	X	X	O	(C99)
fdim [V1.09 or later]	0	X	X	O	(C99)
fdimf [V1.09 or later]	0	X	X	O	(C99)
fdiml [V1.09 or later]	0	X	X	O	(C99)
fmax [V1.09 or later]	0	X	X	O	(C99)
fmaxf [V1.09 or later]	0	X	X	O	(C99)
fmaxl [V1.09 or later]	0	X	X	O	(C99)

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
fmin [V1.09 or later]	0	X	X	O	(C99)
fminf [V1.09 or later]	0	X	X	O	(C99)
fminl [V1.09 or later]	0	X	X	O	(C99)
isgreater [V1.09 or later]	0	X	X	O	(C99)
isgreaterequal [V1.09 or later]	0	X	X	O	(C99)
isless [V1.09 or later]	0	X	X	O	(C99)
islessequal [V1.09 or later]	0	X	X	O	(C99)
islessgreater [V1.09 or later]	0	X	X	O	(C99)
isunordered [V1.09 or later]	0	X	X	O	(C99)
setjmp	0	X	X	Δ	When the pointer reference destination is updated
longjmp	0	X	X	X	SP
va_start	0	X	X	O	
va_arg	0	X	X	O	
va_copy [V1.09 or later]	0	X	X	O	(C99)
va_end	0	X	X	O	
printf	0/43/0/0	X	O	X	stdout, Internal management data
scanf	0/41/0/0	O	X	X	stdin
snprintf [V1.07 or later]	0/43/0/0	X	O	X	Internal management data (C99)
sprintf	0/43/0/0	X	O	X	Internal management data
sscanf	0/41/0/0	X	O	Δ	When the pointer reference destination is updated
vprintf	0/43/0/0	X	O	X	stdout,Internal management data
vscanf [V1.08 or later]	0/41/0/0	O	X	X	stdin (C99)
vsnprintf [V1.07 or later]	0/43/0/0	X	O	X	Internal management data (C99)
vsprintf	0/43/0/0	X	O	X	Internal management data
vsscanf [V1.08 or later]	0/41/0/0	O	X	Δ	When the pointer reference destination is updated (C99)
getchar	0	X	X	X	stdin
gets	0	X	X	X	stdin
_COM_gets_f	0	X	X	X	stdin
putchar	0	X	X	X	stdout
puts	0	X	X	X	stdout
_COM_puts_f	0	X	X	X	stdout
perror	0	O	X	X	errno

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
_COM_perror_f	0	O	X	X	errno
atof	0/41/0/0	O	X	X	errno
_COM_atof_f	0/41/0/0	O	X	X	errno
atoff	0/41/0/0	O	X	X	errno
_COM_atoff_f	0/41/0/0	O	X	X	errno
atoi	0/40/0/0	O	X	X	errno
_COM_atoi_f	0/40/0/0	O	X	X	errno
atol	0/40/0/0	O	X	X	errno
_COM_atol_f	0/40/0/0	O	X	X	errno
atoll [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
_COM_atoll_f [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
strtod	0/41/0/0	O	X	X	errno
_COM_strtod_ff	0/41/0/0	O	X	X	errno
strtof	0/41/0/0	O	X	X	errno
_COM_strtof_ff	0/41/0/0	O	X	X	errno
strtold [V1.07 or later]	0/41/0/0	O	X	X	errno (C99)
_COM_strtold_ff [V1.07 or later]	0/41/0/0	O	X	X	errno (C99)
strtol	0/40/0/0	O	X	X	errno
_COM_strtol_ff	0/40/0/0	O	X	X	errno
strtoll [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
_COM_strtoll_ff [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
strtoul	0/40/0/0	O	X	X	errno
_COM_strtoul_ff	0/40/0/0	O	X	X	errno
strtoull [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
_COM_strtoull_ff [V1.07 or later]	0/43/0/0	O	X	X	errno (C99)
rand	0/24/0/0	O	X	X	seed
srand	0	O	X	X	seed
calloc [V1.02 or later]	0	O	O	X	Internal management data
free [V1.02 or later]	0	O	O	X	Internal management data
malloc [V1.02 or later]	0	O	O	X	Internal management data
realloc [V1.02 or later]	0	O	O	X	Internal management data
abort	0	X	X	-	Processing is not returned

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
bsearch	0	X	X	Δ	When the pointer reference destination is updated
_COM_bsearch_f	0	X	X	Δ	When the pointer reference destination is updated
qsort	0/40/0/0	X	X	Δ	When the pointer reference destination is updated
_COM_qsort_f	0/40/0/0	X	X	Δ	When the pointer reference destination is updated
abs	0	X	X	O	
div	0/39/0/0	X	X	O	
labs	0	X	X	O	
ldiv	0/45/0/0	X	X	O	
llabs [V1.07 or later]	0	X	X	O	(C99)
lldiv [V1.07 or later]	0/42/0/0	X	X	O	(C99)
memcpy	0	X	X	Δ	When the pointer reference destination is updated
_COM_memcpy_f	0	X	X	Δ	When the pointer reference destination is updated
memmove	0	X	X	Δ	When the pointer reference destination is updated
_COM_memmove_ff	0	X	X	Δ	When the pointer reference destination is updated
strcpy	0	X	X	Δ	When the pointer reference destination is updated
_COM_strcpy_ff	0	X	X	Δ	When the pointer reference destination is updated
strncpy	0	X	X	Δ	When the pointer reference destination is updated
_COM_strncpy_ff	0	X	X	Δ	When the pointer reference destination is updated
strcat	0	X	X	Δ	When the pointer reference destination is updated
_COM_strcat_ff	0	X	X	Δ	When the pointer reference destination is updated
strncat	0	X	X	Δ	When the pointer reference destination is updated
_COM_strncat_ff	0	X	X	Δ	When the pointer reference destination is updated
memcmp	0	X	X	Δ	When the pointer reference destination is updated
_COM_memcmp_ff	0	X	X	Δ	When the pointer reference destination is updated

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
strcmp	0	X	X	Δ	When the pointer reference destination is updated
_COM_strcmp_ff	0	X	X	Δ	When the pointer reference destination is updated
strncmp	0	X	X	Δ	When the pointer reference destination is updated
_COM_strncmp_ff	0	X	X	Δ	When the pointer reference destination is updated
memchr	0	X	X	Δ	When the pointer reference destination is updated
_COM_memchr_f	0	X	X	Δ	When the pointer reference destination is updated
strchr	0	X	X	Δ	When the pointer reference destination is updated
_COM_strchr_f	0	X	X	Δ	When the pointer reference destination is updated
strcspn	0	X	X	Δ	When the pointer reference destination is updated
_COM_strcspn_ff	0	X	X	Δ	When the pointer reference destination is updated
strpbrk	0	X	X	Δ	When the pointer reference destination is updated
_COM_strpbrk_ff	0	X	X	Δ	When the pointer reference destination is updated
strrchr	0	X	X	Δ	When the pointer reference destination is updated
_COM_strrchr_f	0	X	X	Δ	When the pointer reference destination is updated
strspn	0	X	X	Δ	When the pointer reference destination is updated
_COM_strspn_ff	0	X	X	Δ	When the pointer reference destination is updated
strstr	0	X	X	Δ	When the pointer reference destination is updated
_COM_strstr_ff	0	X	X	Δ	When the pointer reference destination is updated
strtok	0	X	O	X	Internal management data
memset	0	X	X	Δ	When the pointer reference destination is updated
_COM_memset_f	0	X	X	Δ	When the pointer reference destination is updated
strerror	0	X	X	O	

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
strlen	0	X	X	Δ	When the pointer reference destination is updated
_COM_strlen_f	0	X	X	Δ	When the pointer reference destination is updated
hdwinit	0	X	X	X	Initializing process
stkinit	0	X	X	X	Initializing process

7.6.2 Runtime library

The interrupt disabled time, use of the initialized data section (.data), use of the uninitialized data section (.bss), and the reentrancy of each function in the runtime library are shown in the following.

The interrupt disabled time is displayed, starting from the left, for the case of not using division/multiplication and multiply-accumulate units, the case of using division/multiplication and multiply-accumulate units, the case of using division/multiplication extended instructions (single precision), and the case of using division/multiplication extended instructions (double precision). A single numerical value is common for all libraries.

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
_COM_fadd	0	X	X	O	
_COM_dadd	0	X	X	O	
_COM_fsub	0	X	X	O	
_COM_dsub	0	X	X	O	
_COM_imul	0	X	X	O	
_COM_lmul	0/24/0/0	X	X	O	
_COM_llmul	0/24/0/0	X	X	O	
_COM_fmul	0/14/0/0	X	X	O	
_COM_dmul	0/14/0/0	X	X	O	
_COM_mulsi	0/14/0/0	X	X	O	
_COM_mului	0/14/0/0	X	X	O	
_COM_mulsl	0/14/0/0	X	X	O	
_COM_mulul	0/14/0/0	X	X	O	
_COM_scdiv	0/38/0/0	X	X	O	
_COM_ucdiv	0/38/0/0	X	X	O	
_COM_sidiv	0/38/0/0	X	X	O	
_COM_uidiv	0/38/0/0	X	X	O	
_COM_sldiv	0/40/0/0	X	X	O	
_COM_uldiv	0/40/0/0	X	X	O	
_COM_slldiv	0/43/0/0	X	X	O	
_COM_ulldiv	0/43/0/0	X	X	O	

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
_COM_fdiv	0/41/0/0	X	X	O	
_COM_ddiv	0	X	X	O	
_COM_divui	0/39/0/0	X	X	O	
_COM_divul	0/40/0/0	X	X	O	
_COM_screm	0/38/0/0	X	X	O	
_COM_ucrem	0/38/0/0	X	X	O	
_COM_sirem	0/38/0/0	X	X	O	
_COM_uirem	0/38/0/0	X	X	O	
_COM_slrem	0/40/0/0	X	X	O	
_COM_ulrem	0/40/0/0	X	X	O	
_COM_sllrem	0/42/0/0	X	X	O	
_COM_ullrem	0/42/0/0	X	X	O	
_COM_remui	0/39/0/0	X	X	O	
_COM_remul	0/39/0/0	X	X	O	
_COM_macsi	0/19/16/16	X	X	O	
_COM_macui	0/19/16/16	X	X	O	
_COM_lshl	0	X	X	O	
_COM_llshl	0	X	X	O	
_COM_lshr	0	X	X	O	
_COM_llshr	0	X	X	O	
_COM_lsar	0	X	X	O	
_COM_llsar	0	X	X	O	
_COM_feq	0	X	X	O	
_COM_deq	0	X	X	O	
_COM_fne	0	X	X	O	
_COM_dne	0	X	X	O	
_COM_fge	0	X	X	O	
_COM_dge	0	X	X	O	
_COM_flt	0	X	X	O	
_COM_dlt	0	X	X	O	
_COM_fle	0	X	X	O	
_COM_dle	0	X	X	O	
_COM_fgt	0	X	X	O	
_COM_dgt	0	X	X	O	

Function Name	Interrupt Disabled Time	Use of .data	Use of .bss	Reen-trancy	Remark (non-reentrancy source)
_COM_funordered	0	X	X	O	
_COM_dunordered	0	X	X	O	
_COM_sltof	0	X	X	O	
_COM_sltod	0	X	X	O	
_COM_ultof	0	X	X	O	
_COM_ultod	0	X	X	O	
_COM_slltof	0	X	X	O	
_COM_slltod	0	X	X	O	
_COM_ulltof	0	X	X	O	
_COM_ulltod	0	X	X	O	
_COM_ftosl	0	X	X	O	
_COM_ftoul	0	X	X	O	
_COM_ftosll	0	X	X	O	
_COM_ftoull	0	X	X	O	
_COM_dtosl	0	X	X	O	
_COM_dtoul	0	X	X	O	
_COM_dtosll	0	X	X	O	
_COM_dtoull	0	X	X	O	
_COM_ftod	0	X	X	O	
_COM_dtof	0	X	X	O	
__control_flow_integrity [Professional Edition only] [V1.06 or later]	0	X	X	O	

8. STARTUP

This chapter explains the startup.

8.1 Outline

The startup processing is used to initialize a section for embedding the user application described with the C language to the system or start the main function.
In order to execute a created program, a startup routine appropriate for that program has to be created.

8.2 Startup Routine

Startup routine is the routine that is to be executed after microcontroller is reset and before the execution of main function. Basically, it carries out the initialization after system is reset.
Here describes the following:

- Reset vector table setting
- Register bank setting
- Mirror area setting
- Stack area allocation, stack pointer setting, and stack area initialization
- Initialization of peripheral I/O registers required before main function execution
- Initialization of RAM area section
- Startup of main function
- Creation of termination routine
- Startup of the RL78-S1 core

Some processes are unnecessary depending on the system, and they can be omitted.
Note however that these processes basically need to be written with assembler instructions.

8.2.1 Reset vector table setting

The processing for when a reset (reset input) has been entered is written. In the RL78, upon receiving a reset, a branch is made to the address stored in the address (reset address) that has been determined by the device setting. Therefore, the start address of the startup routine should be set to the reset address. The code will be as follows.

```
_start  .VECTOR 0
```

The start of the startup routine is written as shown below.

```
.SECTION        .text, TEXT
_start:
```

The label of the start of the startup routine is set as "_start" in the example, but another name does not make any difference.

8.2.2 Register bank setting

Since the register bank is initialized to RB0 at a reset, set the register bank as follows only when desiring to set the work register to a register bank other than RB0.
This processing cannot be written for the RL78-S1 core because it does not have the SEL instruction.

```
SEL     RB1
```

8.2.3 Mirror area setting

Since the value of the processor mode control register (PMC) cannot be changed in the RL78-S1 core, this setting is not required. As the PMC value becomes 0 at a reset in the RL78-S2 core or RL78-S3 core, this setting should be made only when the MAA bit value of PMC is to be changed.

For details on the mirror area, PMC, and MAA, see the user's manual of the device.

```
ONEB    !PMC
```

8.2.4 Stack area allocation, stack pointer setting, and stack area initialization

The start (__STACK_ADDR_END) and end (__STACK_ADDR_START) of the stack area are determined by the linker. Since the CC-RL stack area is extended in the direction of address 0x0, set the end (__STACK_ADDR_START) of the stack area used by the system to the stack pointer. For __STACK_ADDR_START and __STACK_ADDR_END, see "6.2.2 Symbols generated by option specifications".

To set the SP in the startup routine, write the code as follows.

```
MOVW    SP,#LOWW(__STACK_ADDR_START)
```

Next, initialize the stack area. The start (__STACK_ADDR_END) of the stack area is passed as an argument. To detect parity errors in response to reading from non-initialized RAM areas, enable the processing for initializing the stack area. If you do not intend to detect parity errors in response to reading from non-initialized RAM areas or the device does not support this facility, the stack area does not need to be initialized. In such cases, comment out the code for initializing the stack area.

```
MOVW    AX,#LOWW(__STACK_ADDR_END)
CALL    !!_stkinit
```

8.2.5 Initialization of peripheral I/O registers required before main function execution

Initialize the peripheral I/O registers which must be set in order to execute the startup routine.

Registers can be set by just writing assembler instructions. They can also be set by branching from the startup routine to a C language function and make the settings from within that C language function. For example, write the following instruction in the startup routine when creating the C language function "void hdwinit(void)" and calling it from the startup routine.

```
CALL    !!_hdwinit
```

8.2.6 Initialization of RAM area section

Copy initial values to "data attribute" areas which are areas with initial values and clear "bss attribute" areas which are areas without initial values to 0. This processing is not required when there is no area that needs to be initialized before program execution.

First, instruct generation of initial value data with the -rom option of the optimizing linker, and define the RAM area section to where data is to be copied. For details, see "8.4 Creating ROM Images".

The defined RAM area section is written to the startup routine as follows.

```
.SECTION        .dataR, DATA
.SECTION        .sdataR, DATA
```

Next, in the startup routine, write a code to initialize the bss attribute areas.
When initializing sections .bss and .sbss to 0, the code will be as follows.

```
        ; clear external variables which doesn't have initial value (near)
        MOVW    HL,#LOWW(STARTOF(.bss))
        MOVW    AX,#LOWW(STARTOF(.bss) + SIZEOF(.bss))
        BR      $.L2_BSS
.L1_BSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_BSS:
        CMPW    AX,HL
        BNZ     $.L1_BSS

        ; clear saddr variables which doesn't have initial value
        MOVW    HL,#LOWW(STARTOF(.sbss))
        MOVW    AX,#LOWW(STARTOF(.sbss) + SIZEOF(.sbss))
        BR      $.L2_SBSS
.L1_SBSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_SBSS:
        CMPW    AX,HL
        BNZ     $.L1_SBSS
```

Then, in the startup routine, write a code to copy data attribute areas to the RAM area section.
When copying sections .data and .sdata to .dataR and .sdataR, respectively, the code will be as follows. Note that the copy routine does not support a section that exceeds the 64-Kbyte boundary.

```
        ; copy external variables having initial value (near)
        MOV     ES,#HIGHW(STARTOF(.data))
        MOVW    BC,#LOWW(SIZEOF(.data))
        BR      $.L2_DATA
.L1_DATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.data))[BC]
        MOV     LOWW(STARTOF(.dataR))[BC],A
.L2_DATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_DATA

        ; copy saddr variables having initial value
        MOV     ES,#HIGHW(STARTOF(.sdata))
        MOVW    BC,#LOWW(SIZEOF(.sdata))
        BR      $.L2_SDATA
.L1_SDATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.sdata))[BC]
        MOV     LOWW(STARTOF(.sdataR))[BC],A
.L2_SDATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_SDATA
```

Write a program for zero-initializing the RAM area section and copying sections in the C language, and this program can be called from the startup routine.

```
        ; initializing RAM section
        CALL    !!_INITSCT_RL
```

A C language example of a function for initializing the RAM area section is given below.

```
#define BSEC_MAX        2           /*Number of BSS sections to be initialized to 0*/
#define DSEC_MAX        2           /*Number of DATA sections to be copied*/

const struct bsec_t {
        char __near *ram_sectop;        /*Start address of section*/
        char __near *ram_secend;        /*End address + 1 of section*/
} bsec_table[BSEC_MAX] = {
        {(char __near *)__sectop(".bss"),
         (char __near *)__secend(".bss")},
        {(char __near *)__sectop(".sbss"),
         (char __near *)__secend(".sbss")}};

const struct dsec_t {
        char __far *rom_sectop;         /*Start address of copy source section*/
        char __far *rom_secend;         /*End address + 1 of copy source section*/
        char __near *ram_sectop;        /*Start address of copy destination section*/
} dsec_table[DSEC_MAX] = {
        {__sectop(".data"),
         __secend(".data"),
         (char __near *)__sectop(".dataR")},
        {__sectop(".sdata"),
         __secend(".sdata"),
         (char __near *)__sectop(".sdataR")}};

void INITSCT_RL(void)
{
        unsigned int i;
        char __far *rom_p;
        char __near *ram_p;

        for (i = 0; i < BSEC_MAX; i++) {
                ram_p = bsec_table[i].ram_sectop;
                for ( ; ram_p != bsec_table[i].ram_secend; ram_p++) {
                        *ram_p = 0;
                }
        }
        for (i = 0; i < DSEC_MAX; i++) {
                rom_p = dsec_table[i].rom_sectop;
                ram_p = dsec_table[i].ram_sectop;
                for ( ; rom_p != dsec_table[i].rom_secend; rom_p++, ram_p++) {
                        *ram_p = *rom_p;
                }
        }
}
```

8.2.6.1 Initialization of RAM area sections by using an initialization table [V1.12 or later]

In V1.12 or later, a table with the information required for initializing RAM area sections can be embedded in an executable file by specifying -ram_init_table_section for the linker.

Each record (row) of the table corresponds to the sections to be initialized and has the following fields.

	Size	Section with initial value	Section without initial value	End record
Field 1	4 bytes	Start address of initial value data	Section start address	0
Field 2	2 bytes	Section size	Section size	0
Field 3	2 bytes	Section start address	Section start address	0

For details, see the description of the link option -ram_init_table_section.

```
> rlink a.obj b.obj -form=absolute -output=a.abs -rom=.data=.dataR
  -rom=.sdata=.sdataR -ram_init_table_section
```

To initialize the RAM by using the table generated by the liker, use the following code instead of the initialization codes for the bss attribute areas and data attribute areas described above.

```
  MOVW DE, #LOWW(STARTOF(.ram_init_table))
  MOV A, #LOW(HIGHW(STARTOF(.ram_init_table)))
  PUSH AX
  PUSH DE

.TABLE_LOOP:
  MOV A, [SP+0x03]
  MOV ES, A
  MOVW AX, [SP+0x00]
  MOVW DE, AX

  CLRB B

  # src_lo
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # src_hi
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # size
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # dest
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  CMP0 B
  BZ $.RETURN

  MOVW AX, DE
  MOVW [SP+0x08], AX
  MOV A, ES
  MOV [SP+0x0B], A
```

```
  # dest
  POP DE
  # size
  POP BC
  # src_hi
  POP AX
  MOV A, X
  MOV ES, A
  # src_lo
  POP HL

  MOVW AX, DE
  CMPW AX, HL
  BNZ $.COPY
  MOV A, ES
  CMP A, #0xF
  BNZ $.COPY

.CLEAR:
  MOVW AX, BC
  OR A, X
  BZ $.TABLE_LOOP
  DECW BC
  CLRB A
  MOV [DE], A
  INCW DE
  BR $.CLEAR

.COPY:
  MOVW AX, BC
  OR A, X
  BZ $.TABLE_LOOP
  DECW BC
  MOV A, ES:[HL]
  MOV [DE], A
  INCW HL
  INCW DE
  BR $.COPY

.RETURN:
  ADDW SP, #0x0C
  RET

.GET_RAM_INIT_RECORD:
  MOVW AX, ES:[DE]
  MOVW HL, AX

  OR A, X
  OR B, A

  MOVW AX, DE
  ADDW AX, #0x0002
  MOVW DE, AX
  MOV A, ES
  ADDC A, #0
  MOV ES, A

  MOVW AX, HL
  RET
```

It is also possible to create an initialization function in C language, and call it in the startup routine. However, take care not to initialize the stack area.

Write as follows in the startup routine:

```
CALL !!_ram_init
```

Initialization function:

```
typedef unsigned char __far * src_ptr;
typedef unsigned short src_len;
typedef unsigned char __near * dest_ptr;
 struct table {
 src_ptr src;
 src_len len;
 dest_ptr dest;
};
typedef struct table __far * table_ptr;

void ram_init(void)
{
 table_ptr record;
 for(record = __sectop(".ram_init_table"); ; record++)
 {
  src_ptr src = record->src;
  src_len len = record->len;
  dest_ptr dest = record->dest;

  if(src == 0 && len == 0 && dest == 0) /* END OF RECORD */
   break;

  if(src == dest)
  {
   /* RAM CLEAR */
   while(len--)
   {
    *dest = 0;
    dest++;
   }
  }
  else
  {
   /* ROM -> RAM COPY */
   while(len--)
   {
    *dest = *src;
    src++;
    dest++;
   }
  }
 }
}
```

8.2.7 Startup of main function

The main function is called when all processings that need to be performed by the startup routine have finished. Write the following instruction to call the main function.

```
CALL    !!_main
```

8.2.8 Creation of termination routine

To go into an infinite loop without processing anything, write the code as follows.

```
_exit:
        BR      $_exit
```

8.2.9 Startup of the RL78-S1 core

In the startup of the RL78-S1 core with a small ROM/RAM area, the entire RAM is initialized instead of initializing the stack area or bss attribute areas, from the point of code efficiency. The start address (__RAM_ADDR_START) and end address (RAM_ADDR_END) of RAM are determined by the linker. For __RAM_ADDR_START and __RAM_ADDR_END, see "6.2.2 Symbols generated by option specifications".

```
        ;-------------------------------------------------
        ; initializing RAM
        ;-------------------------------------------------
        MOVW    HL,#LOWW(__RAM_ADDR_START)
        MOVW    AX,#LOWW(__RAM_ADDR_END)
        BR      $.L2_RAM
.L1_RAM:
        MOV     [HL+0],#0
        INCW    HL
.L2_RAM:
        CMPW    AX,HL
        BNZ     $.L1_RAM
```

8.3 Coding Example

The following is an example of startup routine.

Table 8.1 Examples of startup routine

```
; Copyright (C) 2014 Renesas Electronics Corporation
; RENESAS ELECTRONICS CONFIDENTIAL AND PROPRIETARY.
; This program must be used solely for the purpose for which
; it was furnished by Renesas Electronics Corporation. No part of this
; program may be reproduced or disclosed to others, in any
; form, without the prior written permission of Renesas Electronics
; Corporation.

; NOTE : THIS IS A TYPICAL EXAMPLE

        .public _start
        .public _exit

;------------------------------------------------------------------------------
; RAM section
;------------------------------------------------------------------------------
.SECTION        .dataR, DATA
.SECTION        .sdataR, DATA
```

```
;-----------------------------------------------------------------------
; RESET vector
;-----------------------------------------------------------------------
_start  .VECTOR 0
;-----------------------------------------------------------------------
; startup
;-----------------------------------------------------------------------
.SECTION        .text, TEXT
_start:
        ;------------------------------------------------
        ; setting the stack pointer
        ;------------------------------------------------
        MOVW    SP,#LOWW(__STACK_ADDR_START)

        ;------------------------------------------------
        ; initializing stack area
        ;------------------------------------------------
        MOVW    AX,#LOWW(__STACK_ADDR_END)
        CALL    !!_stkinit

        ;------------------------------------------------
        ; hardware initialization
        ;------------------------------------------------
        CALL    !!_hdwinit
$IFNDEF __USE_RAM_INIT_TABLE

        ;------------------------------------------------
        ; initializing BSS
        ;------------------------------------------------
        ; clear external variables which doesn't have initial value (near)
        MOVW    HL,#LOWW(STARTOF(.bss))
        MOVW    AX,#LOWW(STARTOF(.bss) + SIZEOF(.bss))
        BR      $.L2_BSS
.L1_BSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_BSS:
        CMPW    AX,HL
        BNZ     $.L1_BSS

        ; clear saddr variables which doesn't have initial value
        MOVW    HL,#LOWW(STARTOF(.sbss))
        MOVW    AX,#LOWW(STARTOF(.sbss) + SIZEOF(.sbss))
        BR      $.L2_SBSS
.L1_SBSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_SBSS:
        CMPW    AX,HL
        BNZ     $.L1_SBSS

        ;------------------------------------------------
        ; ROM data copy
        ;------------------------------------------------
        ; copy external variables having initial value (near)
        MOV     ES,#HIGHW(STARTOF(.data))
        MOVW    BC,#LOWW(SIZEOF(.data))
        BR      $.L2_DATA
```

```
.L1_DATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.data))[BC]
        MOV     LOWW(STARTOF(.dataR))[BC],A
.L2_DATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_DATA
        ; copy saddr variables having initial value
        MOV     ES,#HIGHW(STARTOF(.sdata))
        MOVW    BC,#LOWW(SIZEOF(.sdata))
        BR      $.L2_SDATA
.L1_SDATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.sdata))[BC]
        MOV     LOWW(STARTOF(.sdataR))[BC],A
.L2_SDATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_SDATA

$ELSE
        MOVW DE, #LOWW(STARTOF(.ram_init_table))
        MOV A, #LOW(HIGHW(STARTOF(.ram_init_table)))

        CALL $!_ram_init

$ENDIF

        ;-------------------------------------------------
        ; call main function
        ;-------------------------------------------------
        CALL    !!_main          ; main();

        ;-------------------------------------------------
        ; call exit function
        ;-------------------------------------------------
        CLRW    AX               ; exit(0);
_exit:
        BR      $_exit

$IFDEF __USE_RAM_INIT_TABLE
# A,DE: FAR ADDRESS
_ram_init:
        PUSH AX
        PUSH DE

.TABLE_LOOP:
        MOV A, [SP+0x03]
        MOV ES, A
        MOVW AX, [SP+0x00]
        MOVW DE, AX

        CLRB B

        # src_lo
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX
```

```
        # src_hi
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        # size
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        # dest
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        CMP0 B
        BZ $.RETURN

        MOVW AX, DE
        MOVW [SP+0x08], AX
        MOV A, ES
        MOV [SP+0x0B], A

        # dest
        POP DE
        # size
        POP BC
        # src_hi
        POP AX
        MOV A, X
        MOV ES, A
        # src_lo
        POP HL

        MOVW AX, DE
        CMPW AX, HL
        BNZ $.COPY
        MOV A, ES
        CMP A, #0xF
        BNZ $.COPY

.CLEAR:
        MOVW AX, BC
        OR A, X
        BZ $.TABLE_LOOP
        DECW BC
        CLRB A
        MOV [DE], A
        INCW DE
        BR $.CLEAR

.COPY:
        MOVW AX, BC
        OR A, X
        BZ $.TABLE_LOOP
        DECW BC
        MOV A, ES:[HL]
        MOV [DE], A
        INCW HL
        INCW DE
        BR $.COPY
```

```
.RETURN:
        ADDW SP, #0x0C
        RET

.GET_RAM_INIT_RECORD:
        MOVW AX, ES:[DE]
        MOVW HL, AX

        OR A, X
        OR B, A

        MOVW AX, DE
        ADDW AX, #0x0002
        MOVW DE, AX
        MOV A, ES
        ADDC A, #0
        MOV ES, A

        MOVW AX, HL
        RET

$ENDIF

;-------------------------------------------------------------------------------
; section
;-------------------------------------------------------------------------------
.SECTION .RLIB, TEXTF
.L_section_RLIB:
.SECTION .SLIB, TEXTF
.L_section_SLIB:
.SECTION .textf, TEXTF
.L_section_textf:
.SECTION .const, CONST          ; Only for a device with the mirror area
.L_section_const:               ; Only for a device with the mirror area
.SECTION .constf, CONSTF
.L_section_constf:
.SECTION .data, DATA
.L_section_data:
.SECTION .sdata, SDATA
.L_section_sdata:
.SECTION .bss, BSS
.L_section_bss:
.SECTION .sbss, SBSS
.L_section_sbss:
```

The const attribute is not written in the startup routine for a device without the mirror area. [V1.02 or later]

8.4 Creating ROM Images

This section gives an outline of the creation of ROM images that are required for embedded applications.

External and static variables defined in applications are allocated to sections in RAM. If these variables have been initialized, their initial values must be present in RAM when the corresponding application is started.

On the other hand, values in RAM are undefined when the hardware is started up or following a reset. For this reason, the initial values of variables need to have been stored in RAM by the time an application is started after the hardware has been reset.

CC-RL allows the creation of a ROM image that defines how the program code and initial values are allocated to ROM. When the program is run, the initial values in the ROM image are copied to RAM within the startup routine. This initializes the RAM.

Program code in ROM may also have to be placed in RAM in the case of self-programming code. Copying of the program code to RAM is to be done within the startup routine in the same manner as the copying of initial values.

When data are to be copied from ROM to RAM, use the -rom option of the optimizing linker.

```
-rom=name of the initial-value section=name of the destination section
```

When the function to automatically allocate sections is not specified in the linker, the addresses of the initial-value section and destination section need to be specified by the -start option. When the function to automatically allocate sections is specified in the linker, addresses do not have to be specified by the -start option.

```
-start=section-name[,section-name]/destination-address
```

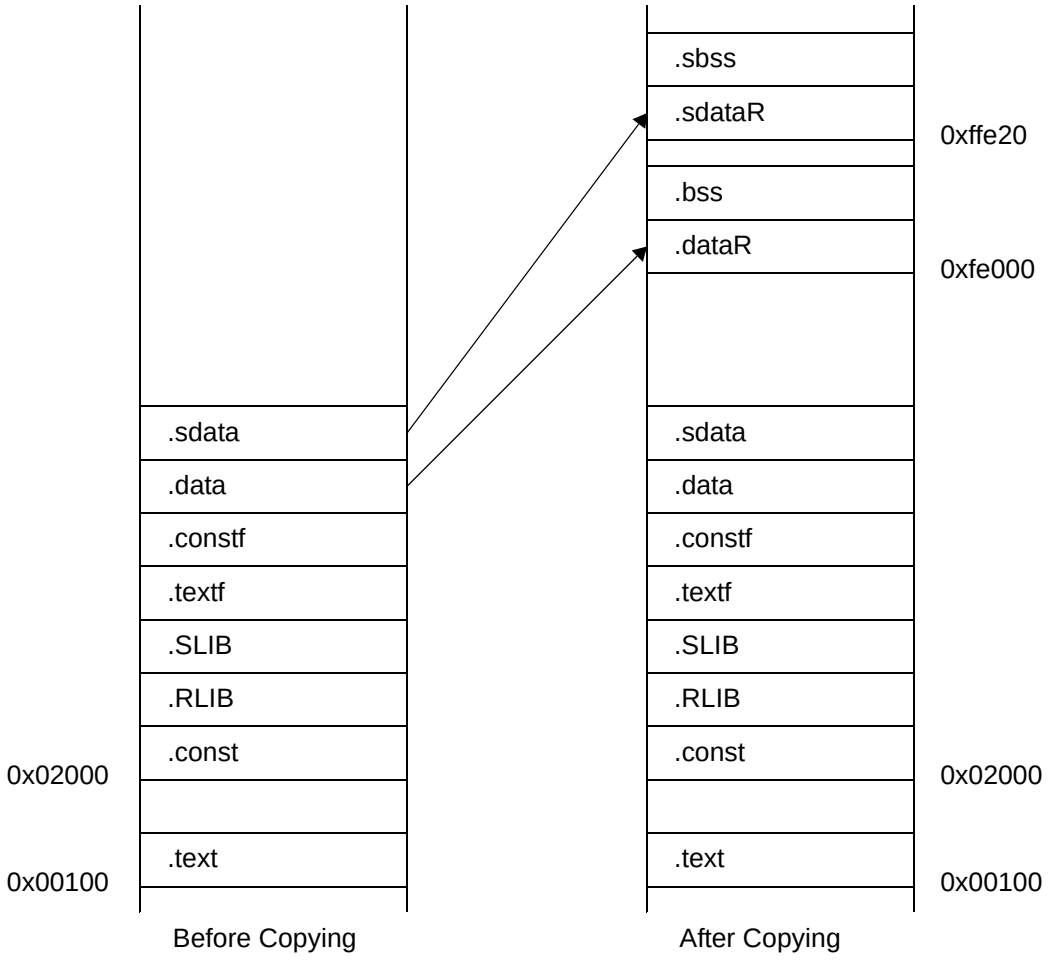
It is assumed that the user program contains sections .text, .textf, .RLIB, .SLIB, .const, .constf, .data, .sdata, .stack_bss, .bss, and .sbss. When locating .text at address 0x100, .const at address 0x2000, reallocation attribute DATA and BSS sections at address 0xfe000, reallocation attribute SDATA and SBSS sections at address 0xffe20 at program execution, the code will be as follows.

```
-start=.text/100
-start=.const,.RLIB,.SLIB,.textf,.constf,.data,.sdata/2000
-start=.dataR,.bss,.stack_bss/FE000
-start=.sdataR,.sbss/FFE20
-rom=.data=.dataR,.sdata=.sdataR
```

The section names .dataR and .sdataR to which data is to be copied can be different names but they need to match the code in the startup routine. At program execution, data is copied from .data to .dataR and from .sdata to .sdataR, and .bss and .sbss are initialized before they are used.

An image of this operation is shown below.

Figure 8.1 Image before and after Copying



9. FUNCTION CALL INTERFACE SPECIFICATIONS

This chapter explains how the CC-RL handles arguments when a function is called.

9.1 Function Call Interface

This section describes the points to be considered in writing an assembly source program when functions in a C source program are called in the assembly source program or vice versa.

9.1.1 General registers and ES/CS registers whose values are guaranteed

This section describes the registers that are guaranteed to have the same values before and after function calls or interrupts, and the registers that are not guaranteed so.

Table 9.1 Relationship between register types and conditions

Register type	Are the Register Values Guaranteed to Be the Same Before and After Function Calls?	Are the Register Values Guaranteed to Be the Same Before and After Interrupts?
General registers AX, BC, DE, and HL	No	Yes
ES and CS registers	No	Yes
PSW and PC registers	No	Yes
MACR register	Yes	Yes
Other registers	No	No

9.1.1.1 General registers AX, BC, DE, and HL

This specification does not guarantee that each of the general registers AX, BC, DE, and HL has the same value before and after function calls.

Only the general registers that belong to the register bank being used are guaranteed to have the same values before and after interrupts. The following two methods can be used to guarantee the values.

- Using a stack area
  Save the register value in a stack area when an interrupt occurs, and restore the value from the stack area when the interrupt ends. This method is also used for registers other than general registers described later.
- Using the register bank changing function
  The register bank specification function of the #pragma directives for interrupt allows you to effectively save and restore general register values by changing the register bank without using a stack area.
  The following describes the cautions about using register bank switching. If these cautions are not observed, the register values are not guaranteed to be the same before and after interrupts.
  - Do not change the register bank within an interrupt handler without using the #pragma directives for interrupt.
  - Do not return from within an interrupt handler by using any method other than the return statement.
  - Do not change to the same register bank as the interrupt source.
    If nested interrupts are used, the first interrupt source and all interrupt handlers up to the return to the source must use different register banks.

9.1.1.2 ES and CS registers

Each of the ES and CS registers are not guaranteed to have the same value before and after function calls.
These registers are guaranteed to have the same values before and after interrupts.

9.1.1.3 PSW and PC registers

Each of the PSW and PC registers are not guaranteed to have the same value before and after function calls.
These registers are guaranteed to have the same values before and after interrupts. Note that the values are automatically saved by hardware when an interrupt occurs. These values are restored by an interrupt end instruction. Therefore, there is no need to explicitly save or restore a value within an interrupt handler.

9.1.1.4 MACR register

The MACR register is guaranteed to have the same value before and after function calls.
This register is also guaranteed to have the same value before and after interrupts.
However, if the -use_mach=not_use compile option is specified (default state in V1.10 or earlier), the compiler generation code does not use[Note] or manage the MACR register. To use the MACR register, you must manage it voluntarily within the program.

Note In this case, some intrinsic functions use the MACR register and rewrite its value.

9.1.1.5 Other registers

Registers other than the above are not guaranteed to have the same value before and after function calls or interrupts.

9.1.2 Passing arguments

(1) Registers used for passing arguments
The registers used for passing arguments are AX, BC, and DE.

(2) Target for argument allocation, argument type, and size change
The targets for allocating arguments, argument types, and how to change the size are shown below.
Changes can be accepted only for those described below.

Argument Class	Target for Argument Allocation	Argument Type and Size Change
When the function prototype can be referenced and the parameter type can be referenced	Register or stack	When allocating a far pointer argument to registers, it is allocated to registers for three bytes.
When the function prototype can be referenced but the parameter type cannot be referenced (= variable argument)	Stack	Argument type conforms to default argument promotions.
When the function prototype cannot be referenced	Register or stack	Argument type conforms to default argument promotions. When allocating a far pointer argument to registers as the result of default argument promotions, it is allocated to registers for three bytes.

Caution 1. For the variable size or default argument promotions, see "(7) Default argument promotions".

Caution 2. The argument immediately before a variable argument is handled as "when the function prototype can be referenced and the parameter type can be referenced".

(3) Register allocation of arguments
Arguments are allocated to registers as follows.

- A target to be allocated to registers is an argument whose size is 4 bytes or less.
  Note that when an argument of the structure type or union type is to be allocated to registers, the included padding will also be allocated to registers.
- When the argument is a structure or union, all members are allocated either to registers or in the stack.
- For a far pointer, the lower three bytes are allocated to registers.
  The page number part (= upper four bits of the 20-bit far address) of a far pointer is stored in the lower four bits of a register to which the upper one byte among the three bytes will be allocated.
  For example, when allocating a far pointer to registers A-DE, the page number part is stored in the lower four bits of register A.

- Register allocation is processed in the order from the first argument to the last argument (from left to right in the source program), and allocates each argument to the register with the highest priority among available registers. If there are no available registers, arguments are allocated in the stack. The priorities of the registers are shown below.

Argument Size	Priority of Register to which Argument is Allocated (left side has the highest priority)
1-byte	A, X, C, B, E, D
2-byte	AX, BC, DE
3-byte	C-AX, X-BC, E-BC, X-DE, B-DE
4-byte	far pointer: A-DE, X-DE, C-DE, B-DE,X-BC Other than far pointer: BC-AX, DE-BC

"-" in this table is a symbol to associate 8-bit or 16-bit registers to other 16-bit registers.
Each argument is allocated to registers so that the descending order of addresses (from upper address to lower address) for the bytes composing the argument matches the register specification order (from left to right) shown in the above table.

Example 1. When calling to a function declared as "void foo (char p1, short p2, char p3)", p1 is allocated to register A, p2 is allocated to register BC, and p3 is allocated to register X.

Example 2. When structure-type argument S shown below is allocated to registers, c1 is allocated to register X and s2 is allocated to register BC. Padding is allocated to register A.

```
struct {
        char    c1;
        short   s2;
} S;
```

Example 3. When calling to a function declared as "void foo (long x)", the upper 2 bytes of x are allocated to register BC and the lower 2 bytes are allocated to register AX.

Example 4. When structure-type argument S3 shown below is allocated to registers, the highest byte is allocated to register C, the next one byte is allocated to register A, and the lowest byte is allocated to register X.

```
struct{
        char    a[3];
} S3;
```

Caution 1. 8-byte data, such as long long type data or double type data in the program to which option -dbl_size=8 is specified, is allocated in the stack

Caution 2. A structure or union of five bytes or more is allocated in the stack while a structure or union of 4 bytes or less is a target for being allocated to registers.

(4) Stack allocation of arguments
Arguments are allocated in the stack as follows.

- Arguments to be passed by the stack are allocated in little endian mode and aligned at the 2-byte boundary.
- The order for allocating the arguments to be passed by the stack is that the more the argument is on the left side in the argument sequence, the address is smaller.
- The arguments to be passed by the stack are sequentially allocated in the stack except for the 1-byte padding which can be inserted between arguments.
- Each argument is allocated in the stack so that the descending order of addresses (from upper address to lower address) for the bytes composing the argument matches the descending order of addresses (from upper address to lower address) in the stack.
- For a far pointer, the lower three bytes are allocated to a 4-byte area in the stack. The value of the highest one byte of the 4 bytes is undefined. The page number part (= upper four bits of the 20-bit far address) of a far pointer is stored in the lower four bits of the second upper byte in the 4-byte area.

Caution For the method of stack allocation, see section "9.1.4 Stack frame".

- The arguments to be allocated to the stack are as follows.
  - Argument whose size is between one byte and 4 bytes and is not allocated to registers
  - Argument whose size is five bytes or more
  - Variable argument

Example When calling to a function declared as "void foo (long long x)" and assuming that sp indicates the value of stack pointer immediately before the call site of the function, the highest one byte of x is allocated to the location indicated by sp + 7, each byte is allocated to sp + 6, sp + 5, sp + 4, sp + 3, sp + 2, and sp + 1 in the descending order of addresses, and the lowest byte is allocated to the location indicated by sp + 0.

9.1.3 Return value

(1) Registers used for passing of return values
The registers used for passing of return values are AX, BC, and DE.

(2) Allocation of return values and pointers to return values to registers
Return values and pointers to return values are allocated to registers as follows.

- For a return value with a size of five bytes or more, the pointer to the return value is set to the first parameter as a near pointer.
  This means that the passing conventions for the first parameter are applied to the return value. In accordance with this, the passing conventions for (n + 1)-th parameter are applied to n-th parameter (n = 1, ..., N) which is specified in the program
  For the details of description, see "9.1.2 Passing arguments".
- A return value whose size is 4 bytes or less is allocated to registers.
- For the method of allocating a far pointer to registers, see section "9.1.2 Passing arguments".
- A 4-byte structure or union whose only member is a far pointer is considered to be other than a far pointer.
- The conventions for allocating a return value of 4 bytes or less to registers are given in the following table.

Return Value Size	Register to which Argument is Allocated
1-byte	A
2-byte	AX
3-byte	C-AX
4-byte	far pointer: A-DE Other than far pointer: BC-AX

"-" in this table is a symbol to associate 8-bit or 16-bit registers to other 16-bit registers.
Each return value is allocated to registers so that the descending order of addresses (from upper address to lower address) for the bytes composing the return value matches the register specification order (from left to right) shown in the above table.

Caution Structures or unions of 4 bytes or less are targets to be allocated to registers.

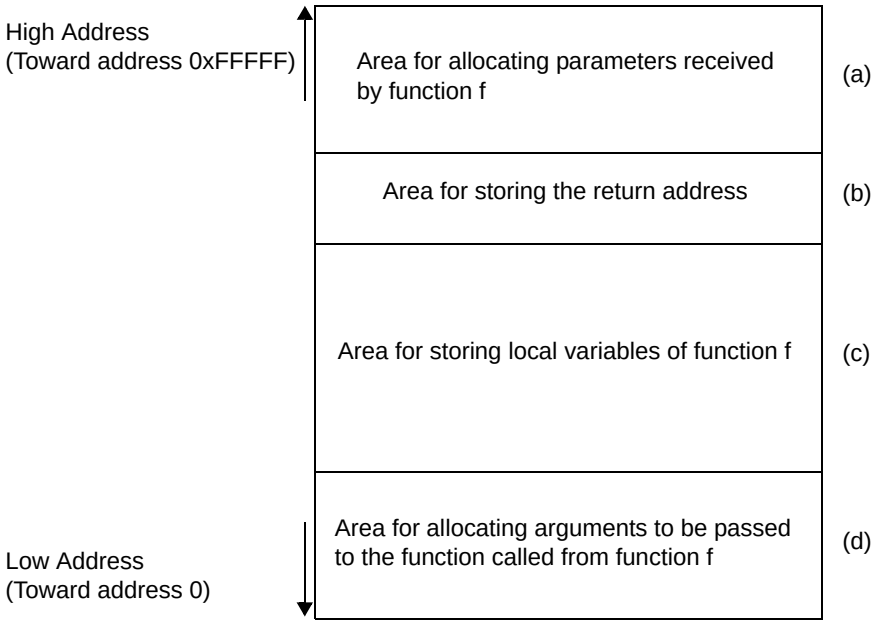
9.1.4 Stack frame

(1) Value to be set to stack pointer
An address which is a multiple of 2 is set to the stack pointer (SP).

(2) Allocation and deallocation of stack frame
The stack pointer always points to the lowest address of the stack frame. Therefore, values stored in areas with smaller addresses than the SP address are not guaranteed.
A stack area allocated by the caller function when the function is called is deallocated by the caller function on returning from the function call. Accordingly, the SP indicates the return address storage area at the entrance and exit of the callee function.

(3) Structure of the stack frame
Below is shown the stack frame of functions f and g from the perspective of function f (Callee function), when function f is called by function g (Caller function).

Figure 9.1 Contents of Stack Frame



Below is the area that function f can reference and set.

(a) Area for allocating parameters received by function f
This is an area for setting parameters that are not allocated to registers. The data set in this area is aligned to the 2-byte boundary. When all parameters are allocated to registers, the size of this area becomes 0.

(b) Area for storing the return address
This is an area for allocating the return address. The size of this area is fixed to 4 bytes, and the address is set as a far address.
The value of the upper one byte of the 4 bytes is undefined.

(c) Area for storing local variables of function f
This is a stack area used by function f to store local variables.

(d) Area for allocating arguments to be passed to the function called from function f
This is an area for setting arguments to be allocated to the stack when function f calls another function. If all arguments required for the function call are allocated to registers, the size of this area becomes 0.

9.2 Calling of Assembly Language Routine from C Language

This section explains the points to be noted when calling an assembler function from a C function.

(1) Identifier
If external names, such as functions and external variables, are described in the C source, the CC-RL adds them prefix "_" (underscore) in the assembly program.

Table 9.2 Identifier

C	Assembly Program
func1 ( )	_func1

Add prefix "_" to the identifier when defining functions and external variables with the assembler and remove "_" when referencing them from a C function.

(2) Stack frame
The CC-RL generates codes on the assumption that the stack pointer (SP) always indicates the lowest address of the stack frame. Therefore, the address area lower than the address indicated by SP can be freely used in the

assembler function after branching from a C function to an assembler function. Conversely, if the contents of the higher address area are changed, the area used by a C function may be lost and the subsequent operation cannot be guaranteed. To avoid this, change SP at the beginning of the assembler function before using the stack.
At this time, furthermore, make sure that the value of SP becomes equal both at the beginning and the end of the assembler function.
Registers can be used freely in assembler functions (register values do not have to be saved before usage and restored after usage).

(3) Return address passed to C function
The CC-RL generates codes on the assumption that the return address of a function is stored at the top of the stack. When execution branches to an assembler function, the return address of the function is stored at the top of the stack. Execute the ret instruction to return to a C function.

(4) Prototype declaration of assembler function
In the CC-RL, a prototype declaration is necessary for functions called from a C function. Make a correct prototype declaration even for assembler functions.

9.3 Calling of C Language Routine from Assembly Language

This section explains the points to be noted when calling a C function from an assembler function.

(1) Stack frame
The CC-RL generates codes on the assumption that the stack pointer (SP) always indicates the lowest address of the stack frame. Therefore, set SP so that it indicates the higher address of an unused area of the stack area before branching from an assembler function to a C function. This is because the stack frame is allocated towards the lower addresses.

(2) Register
The CC-RL does not guarantee that registers have the same values before and after a C function is called, except for some registers. Therefore, do not leave a value that must be retained assigned to a register.

(3) Return address returned to assembler function
The CC-RL generates codes on the assumption that the return address of a function is stored at the top of the stack. When branching to a C function in an assembly program, the return address of the function must be stored at the top of the stack.
Generally, use a call instruction for branching to C functions, which makes the return address of the function stored at the top of the stack.

9.4 Reference of Argument Defined by Other Language

The method of referring to the variable defined by the assembly language on the C language is shown below.

Example Programming of C Language

```
extern  char    c;
extern  int     i;

void subf() {
        c = 'A';
        i = 4;
}
```

The CC-RL assembler performs as follows.

```
        .PUBLIC _i
        .PUBLIC _c
        .SECTION        .data, DATA
_i:
        .DB4    0x0
_c:
        .DB     0x0
```

10. MESSAGE

This chapter describes message that CC-RL outputs.

10.1 General

This section describes internal error message, error message, fatal error message, information message and warning message that CC-RL outputs.

10.2 Message Formats

This section describes the output formats of messages.
The output formats of messages are as follows.

(1) When the file name and line number are included

```
file-name (line-number) : message-type 05 message-number : message
```

(2) When the file name and line number are not included

```
message-type 05 message-number : message
```

Remark Following contents are output as the continued character string.
Message Types : 1 alphabetic character
Message Numbers : 5 digits

10.3 Message Types

This section describes the message types displayed by CC-RL.
The message types (1 alphabetic character) are as follows.

Table 10.1 Message Type

Message Type	Description
C	Internal error : Processing is aborted. No object codes are generated.
E	Error : Processing is aborted if a set number of errors occur. No object codes are generated.
F	Fatal error : Processing is aborted. No object codes are generated.
M	Information : Processing continues. Object codes are generated.
W	Warning : Processing continues. Object codes are generated (They might not be what the user intended).

10.4 Message Numbers

This section describes the message numbers displayed by CC-RL.
The message numbers when the CC-RL is executed are 5 digits number output following number (05).

10.5 Messages

This section describes the messages displayed by CC-RL.

10.5.1 Internal errors

Table 10.2 Internal Errors

C05*nnnnn*	[Message]	Internal error (*information*).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0511200	[Message]	Internal error(*error-information*).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0519996	[Message]	Out of memory.
	[Explanation]	The amount of data input (source file name and specified options) to the ccrl command is too large.
	[Action by User]	Divide the data input to the ccrl command, and then perform startup several times.
C0519997	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0520000	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0529000	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530001	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530002	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530003	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530004	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530005	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0530006	[Message]	Internal Error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0550802	[Message]	Internal error(action type of icode strage).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0550804	[Message]	Internal error(section name ptr not found(*string*)).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0550805	[Message]	Internal error(section list ptr not found(*string*)).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0550806	[Message]	Internal error(current section ptr not found(*string*)).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.

C0550808	[Message]	Internal error(*string*).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0551800	[Message]	Internal error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0564000	[Message]	Internal error : ("*internal error number*") "*file line number*" / "*comment*"
	[Explanation]	An internal error occurred during processing by the optimizing linker.
	[Action by User]	Make a note of the internal error number, file name, line number, and comment in the message, and contact the support department of the vendor.
C0564001	[Message]	Internal error
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0580013	[Message]	internal error. unexpected syntax error message.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580014	[Message]	internal error. section "%s" is not found.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580015	[Message]	internal error. function for "%s" is not found.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580016	[Message]	internal error. additional bytes are invalid "%s".
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580900	[Message]	internal error. undefined message for "%s".
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580901	[Message]	internal error. failed to set signal handler.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580902	[Message]	internal error. invalid memory access.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580903	[Message]	internal error. invalid instruction execution.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580904	[Message]	internal error. abnormal termination.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580905	[Message]	internal error. illegal operation.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580906	[Message]	internal error. unexpected signal received.
	[Action by User]	Contact your vendor or Renesas Electronics.
C0580907	[Message]	internal error. unknown message type "%s".
	[Action by User]	Contact your vendor or Renesas Electronics.
C0590001	[Message]	Internal error
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.

10.5.2 Errors

Table 10.3 Errors

E0511101	[Message]	"*path*" specified by the "*character string*" option is a folder. Specify an input file.
E0511102	[Message]	The file "*file*" specified by the "*character string*" option is not found.
E0511103	[Message]	"*path*" specified by the "*character string*" option is a folder. Specify an output file.
E0511104	[Message]	The output folder "*folder*" specified by the "*character string*" option is not found.
E0511107	[Message]	"*path*" specified by the "*character string*" option is not found.
	[Explanation]	"*path*" (*file-name* or *folder*) specified in the "*character string*" option was not found.
E0511108	[Message]	The "*character string*" option is not recognized.
E0511109	[Message]	The "*character string*" option can not have an argument.
E0511110	[Message]	The "*character string*" option requires an argument.
	[Explanation]	The "*character string*" option requires an argument. Specify the argument.
E0511113	[Message]	Invalid argument for the "*character string*" option.
E0511114	[Message]	Invalid argument for the "-O*character string*" option.
E0511115	[Message]	The "-O*character string*" option is invalid.
E0511116	[Message]	The "-O*character string*" option is not recognized.
E0511117	[Message]	Invalid parameter for the "*character string*" option.
E0511121	[Message]	Multiple source files are not allowed when both the "-o" option and the "*character string*" option are specified.
E0511124	[Message]	The "-cpu" option must be specified.
E0511129	[Message]	Command file "*file*" is read more than once.
E0511130	[Message]	Command file "*file*" cannot be read.
E0511131	[Message]	Syntax error in command file "*file*".
E0511133	[Message]	The parameter for the "*character string*" option must be a folder when multiple source files are specified.
E0511134	[Message]	Input file "*file*" is not found.
E0511135	[Message]	"*path*" specified as an input file is a folder.
E0511145	[Message]	"*character string2*" specified in the "*character string1*" option is not available.
E0511150	[Message]	The "*character string1*" option and the "*character string2*" option are inconsistent.
E0511152	[Message]	The "*character string1*" option needs the "*character string2*" option.
E0511154	[Message]	Component file "*file name*" for the *compiler package name* is not found. Reinstall the *compiler package name*.
E0511177	[Message]	"*character string*" is specified more than once.
E0511178	[Message]	"*character string*" option is unavailable because the license of *version* Professional edition is not found. Please consider purchasing the product of Professional edition.
E0511182	[Message]	File access error.(*information*)

E0520001	[Message]	Last line of file ends without a newline.
	[Action by User]	The last line in the file does not end with a line break. Add a line break.
E0520005	[Message]	Could not open source file "*file name*".
E0520006	[Message]	Comment unclosed at end of file.
	[Action by User]	There is an unclosed comment at the end of the file. Make sure that there are no unclosed comments.
E0520007	[Message]	Unrecognized token.
	[Action by User]	Unknown token. Check the indicated location.
E0520008	[Message]	Missing closing quote.
	[Action by User]	The string is missing a closing quotation mark. Make sure that there are no unclosed quotation mark.
E0520010	[Message]	"#" not expected here.
	[Explanation]	There is a "#" character in an invalid location.
E0520011	[Message]	Unrecognized preprocessing directive.
E0520012	[Message]	Parsing restarts here after previous syntax error.
E0520013	[Message]	Expected a file name.
E0520014	[Message]	Extra text after expected end of preprocessing directive.
	[Action by User]	The iodefine.h file is generated when a project is created in an integrated development environment. Since the interrupt request names are defined in this file, iodefine.h should be included before #pragma directives if the interrupt request names have been written. Note that file inclusion can also be specified by the -preinclude option of the compiler.
E0520017	[Message]	Expected a "]".
E0520018	[Message]	Expected a ")".
E0520019	[Message]	Extra text after expected end of number.
E0520020	[Message]	Identifier "*character string*" is undefined.
	[Action by User]	The iodefine.h file is generated when a project is created in an integrated development environment. When a reserved word of SFR is used, this file must be included. PSW cannot be accessed directly. To manipulate PSW, use the intrinsic function __get_psw or __set_psw. To perform bit access, use the type defined in iodefine.h.
E0520022	[Message]	Invalid hexadecimal number.
E0520023	[Message]	Integer constant is too large.
E0520024	[Message]	Invalid octal digit.
	[Explanation]	Invalid hexadecimal number. Hexadecimal numbers cannot contain '8' or '9'.
E0520025	[Message]	Quoted string should contain at least one character.
E0520026	[Message]	Too many characters in character constant.
E0520027	[Message]	Character value is out of range.
E0520028	[Message]	Expression must have a constant value.
E0520029	[Message]	Expected an expression.

E0520030	[Message]	Floating constant is out of range.
E0520031	[Message]	Expression must have integral type.
E0520032	[Message]	Expression must have arithmetic type.
E0520033	[Message]	Expected a line number
	[Explanation]	The line number after the "#line" statement does not exist.
E0520034	[Message]	Invalid line number
	[Explanation]	The line number after the "#line" statement is invalid.
E0520036	[Message]	The #if for this directive is missing.
E0520037	[Message]	The #endif for this directive is missing.
E0520038	[Message]	Directive is not allowed -- an #else has already appeared.
	[Explanation]	This directive is invalid because there is already an "#else" statement.
E0520039	[Message]	Division by zero.
E0520040	[Message]	Expected an identifier.
E0520041	[Message]	Expression must have arithmetic or pointer type.
E0520042	[Message]	Operand types are incompatible ("*type1*" and "*type2*").
E0520044	[Message]	Expression must have pointer type.
E0520045	[Message]	#undef may not be used on this predefined name.
E0520046	[Message]	"*macro*" is predefined; attempted redefinition ignored.
	[Explanation]	The macro "*macro*" is predefined. It cannot be redefined.
E0520047	[Message]	Incompatible redefinition of macro "*macro*" (declared at line *number*).
	[Explanation]	The redefinition of macro "*macro*" is not compatible with the definition at line number.
E0520049	[Message]	Duplicate macro parameter name.
E0520050	[Message]	"##" may not be first in a macro definition.
E0520051	[Message]	"##" may not be last in a macro definition.
E0520052	[Message]	Expected a macro parameter name.
E0520053	[Message]	Expected a ":".
E0520054	[Message]	Too few arguments in macro invocation.
E0520055	[Message]	Too many arguments in macro invocation.
E0520056	[Message]	Operand of sizeof may not be a function.
E0520057	[Message]	This operator is not allowed in a constant expression.
E0520058	[Message]	This operator is not allowed in a preprocessing expression.
E0520059	[Message]	Function call is not allowed in a constant expression.
E0520060	[Message]	This operator is not allowed in an integral constant expression.
E0520061	[Message]	Integer operation result is out of range.
E0520062	[Message]	Shift count is negative.
E0520063	[Message]	Shift count is too large.

E0520064	[Message]	Declaration does not declare anything.
E0520065	[Message]	Expected a ";".
	[Action by User]	The iodefine.h file is generated when a project is created in an integrated development environment. When a reserved word of SFR is used, this file must be included. PSW cannot be accessed directly. To manipulate PSW, use the intrinsic function __get_psw or __set_psw. To perform bit access, use the type defined in iodefine.h.
E0520066	[Message]	Enumeration value is out of "int" range.
E0520067	[Message]	Expected a "}".
E0520069	[Message]	Integer conversion resulted in truncation.
E0520070	[Message]	Incomplete type is not allowed.
E0520071	[Message]	Operand of sizeof may not be a bit field.
E0520075	[Message]	Operand of "*" must be a pointer.
E0520077	[Message]	This declaration has no storage class or type specifier.
E0520078	[Message]	A parameter declaration may not have an initializer.
E0520079	[Message]	Expected a type specifier.
E0520080	[Message]	A storage class may not be specified here.
E0520081	[Message]	More than one storage class may not be specified.
	[Explanation]	Multiple storage class areas have been specified. Only one storage class area can be specified.
E0520083	[Message]	Type qualifier specified more than once.
	[Explanation]	Multiple type qualifiers have been specified. It is not possible to specify more than one type qualifier.
E0520084	[Message]	Invalid combination of type specifiers.
E0520085	[Message]	Invalid storage class for a parameter.
E0520086	[Message]	Invalid storage class for a function.
E0520087	[Message]	A type specifier may not be used here.
E0520088	[Message]	Array of functions is not allowed.
E0520089	[Message]	Array of void is not allowed.
E0520090	[Message]	Function returning function is not allowed.
E0520091	[Message]	Function returning array is not allowed.
E0520092	[Message]	Identifier-list parameters may only be used in a function definition.
E0520093	[Message]	Function type may not come from a typedef.
E0520094	[Message]	The size of an array must be greater than zero.
E0520095	[Message]	Array is too large.
E0520097	[Message]	A function may not return a value of this type.
E0520098	[Message]	An array may not have elements of this type.
E0520099	[Message]	A declaration here must declare a parameter.
E0520100	[Message]	Duplicate parameter name.

E0520101	[Message]	"*symbol*" has already been declared in the current scope.
E0520102	[Message]	Forward declaration of enum type is nonstandard.
E0520104	[Message]	Struct or union is too large.
E0520105	[Message]	Invalid size for bit field.
E0520106	[Message]	Invalid type for a bit field.
E0520107	[Message]	Zero-length bit field must be unnamed.
E0520109	[Message]	Expression must have (pointer-to-) function type.
E0520110	[Message]	Expected either a definition or a tag name.
E0520112	[Message]	Expected "while".
E0520114	[Message]	Type "*symbol*" was referenced but not defined.
E0520115	[Message]	A continue statement may only be used within a loop.
E0520116	[Message]	A break statement may only be used within a loop or switch.
E0520117	[Message]	Non-void "*function name*" should return a value.
E0520118	[Message]	A void function may not return a value.
E0520119	[Message]	Cast to type "*type*" is not allowed.
E0520120	[Message]	Return value type does not match the function type.
E0520121	[Message]	A case label may only be used within a switch.
E0520122	[Message]	A default label may only be used within a switch.
E0520124	[Message]	default label has already appeared in this switch.
E0520125	[Message]	Expected a "(".
E0520127	[Message]	Expected a statement.
E0520129	[Message]	A block-scope function may only have extern storage class.
E0520130	[Message]	Expected a "{".
E0520132	[Message]	Expression must have pointer-to-struct-or-union type.
E0520134	[Message]	Expected a field name.
E0520136	[Message]	*Type* "*symbol*" has no field "*field*".
E0520137	[Message]	Expression must be a modifiable lvalue.
E0520138	[Message]	Taking the address of a register variable is not allowed.
E0520139	[Message]	Taking the address of a bit field is not allowed.
E0520140	[Message]	Too many arguments in function call.
E0520141	[Message]	Unnamed prototyped parameters not allowed when body is present.
E0520142	[Message]	Expression must have pointer-to-object type.
E0520144	[Message]	A value of type "*type1*" cannot be used to initialize an entity of type "*type2*".
E0520145	[Message]	*Type* "*symbol*" may not be initialized.
E0520146	[Message]	Too many initializer values.
E0520147	[Message]	Declaration is incompatible with "*declaration*" (declared at line *number*).

E0520148	[Message]	Tyep "*symbol*" has already been initialized.
E0520149	[Message]	A global-scope declaration may not have this storage class.
E0520151	[Message]	A typedef name may not be redeclared as a parameter.
E0520154	[Message]	Expression must have struct or union type.
E0520158	[Message]	Expression must be an lvalue or a function designator.
E0520159	[Message]	Declaration is incompatible with previous "*declaration*" (declared at line *number*).
E0520165	[Message]	Too few arguments in function call.
E0520166	[Message]	Invalid floating constant.
E0520167	[Message]	Argument of type "*type1*" is incompatible with parameter of type "*type2*".
E0520168	[Message]	A function type is not allowed here.
E0520169	[Message]	Expected a declaration.
E0520171	[Message]	Invalid type conversion.
E0520172	[Message]	External/internal linkage conflict with previous declaration.
E0520173	[Message]	Floating-point value does not fit in required integral type.
E0520175	[Message]	Subscript out of range.
E0520179	[Message]	Right operand of "%" is zero.
E0520183	[Message]	Type of cast must be integral.
E0520184	[Message]	Type of cast must be arithmetic or pointer.
E0520221	[Message]	Floating-point value does not fit in required floating-point type.
E0520222	[Message]	Floating-point operation result is out of range.
E0520223	[Message]	Function *xxx* declared implicitly.
E0520228	[Message]	Trailing comma is nonstandard.
	[Explanation]	A trailing comma is not standard.
E0520235	[Message]	Variable any-string was declared with a never-completed type.
E0520238	[Message]	Invalid specifier on a parameter.
E0520240	[Message]	Duplicate specifier in declaration.
E0520247	[Message]	*Type* "*symbol*" has already been defined.
E0520253	[Message]	Expected a ",".
E0520254	[Message]	Type name is not allowed.
E0520256	[Message]	Invalid redeclaration of type name "*type*".
	[Explanation]	Type name "*type*" was redeclared illegally.
E0520260	[Message]	Explicit type is missing ("int" assumed).
E0520267	[Message]	Old-style parameter list (anachronism).
E0520268	[Message]	Declaration may not appear after executable statement in block.
E0520274	[Message]	Improperly terminated macro invocation.
E0520284	[Message]	NULL reference is not allowed.

E0520296	[Message]	Invalid use of non-lvalue array.
E0520301	[Message]	typedef name has already been declared (with same type).
E0520313	[Message]	Type qualifier is not allowed on this function.
E0520325	[Message]	inline specifier allowed on function declarations only.
E0520340	[Message]	Value copied to temporary, reference to temporary used.
E0520375	[Message]	Declaration requires a typedef name.
E0520393	[Message]	Pointer to incomplete class type is not allowed.
E0520401	[Message]	Destructor for base class *type* is not virtual.
E0520404	[Message]	Function "main" may not be declared inline.
E0520409	[Message]	*Type* "*symbol*" returns incomplete type "*type*".
E0520411	[Message]	A parameter is not allowed.
E0520445	[Message]	*name1* is not used in declaring the parameter types of "*name2*".
E0520450	[Message]	The type "long long" is nonstandard.
E0520451	[Message]	Omission of "class" is nonstandard.
E0520460	[Message]	declaration of *xxx* hides function parameter.
E0520469	[Message]	Tag kind of *xxx* is incompatible with declaration of "symbol".
E0520490	[Message]	*name* cannot be instantiated -- it has been explicitly specialized.
E0520494	[Message]	Declaring a void parameter list with a typedef is nonstandard.
E0520513	[Message]	A value of type "type1" cannot be assigned to an entity of type "type2".
E0520520	[Message]	Initialization with "{...}" expected for aggregate object.
E0520521	[Message]	Pointer-to-member selection class types are incompatible (*type1* and *type2*).
E0520525	[Message]	A dependent statement may not be a declaration.
	[Explanation]	Cannot write declaration due to lack of "{" character after "if()" statement.
E0520526	[Message]	A parameter may not have void type.
E0520545	[Message]	Use of a local type to declare a function.
E0520560	[Message]	symbol is reserved for future use as a keyword.
E0520561	[Message]	Invalid macro definition:
E0520562	[Message]	Invalid macro undefinition:
E0520606	[Message]	This pragma must immediately precede a declaration.
E0520607	[Message]	This pragma must immediately precede a statement.
E0520608	[Message]	This pragma must immediately precede a declaration or statement.
E0520609	[Message]	This kind of pragma may not be used here.
E0520618	[Message]	struct or union declares no named members.
E0520619	[Message]	Nonstandard unnamed field.
E0520643	[Message]	"restrict" is not allowed.
E0520644	[Message]	A pointer or reference to function type may not be qualified by "restrict".

E0520654	[Message]	Declaration modifiers are incompatible with previous declaration.
E0520655	[Message]	The modifier *NAME* is not allowed on this declaration.
E0520660	[Message]	Invalid packing alignment value.
E0520676	[Message]	Using out-of-scope declaration of *type* "*symbol*" (declared at line *number*).
E0520702	[Message]	Expected an "=".
E0520705	[Message]	Default template arguments are not allowed for function templates.
E0520731	[Message]	Array with incomplete element type is nonstandard.
E0520732	[Message]	Allocation operator may not be declared in a namespace.
E0520733	[Message]	Deallocation operator may not be declared in a namespace.
E0520744	[Message]	Incompatible memory attributes specified.
E0520747	[Message]	Memory attribute specified more than once.
E0520749	[Message]	A type qualifier is not allowed.
E0520757	[Message]	*NAME* is not a type name.
E0520761	[Message]	typename may only be used within a template.
E0520765	[Message]	Nonstandard character at start of object-like macro definition.
E0520766	[Message]	Exception specification for virtual *name1* is incompatible with that of overridden *name2*.
E0520767	[Message]	Conversion from pointer to smaller integer.
E0520768	[Message]	Exception specification for implicitly declared virtual *name1* is incompatible with that of overridden *name2*.
E0520784	[Message]	A storage class is not allowed in a friend declaration.
E0520793	[Message]	Explicit specialization of %n must precede its first use.
E0520811	[Message]	const *name* requires an initializer -- class *type* has no explicitly declared default constructor.
E0520816	[Message]	In a function definition a type qualifier on a "void" return type is not allowed.
E0520833	[Message]	Pointer or reference to incomplete type is not allowed.
E0520845	[Message]	This partial specialization would have been used to instantiate *name*.
E0520846	[Message]	This partial specialization would have made the instantiation of *name* ambiguous.
E0520852	[Message]	Expression must be a pointer to a complete object type.
E0520858	[Message]	*name* is a pure virtual function.
E0520859	[Message]	Pure virtual *name* has no overrider.
E0520861	[Message]	Invalid character in input line.
E0520862	[Message]	Function returns incomplete type "*type*".
E0520870	[Message]	Invalid multibyte character sequence.
E0520886	[Message]	Invalid suffix on integral constant.
	[Explanation]	The integer constant has an invalid suffix.
E0520935	[Message]	Typedef may not be specified here.

E0520938	[Message]	Return type "int" omitted in declaration of function "main".
E0520940	[Message]	Missing return statement at end of non-void *type* "*symbol*".
E0520946	[Message]	Name following "template" must be a template.
E0520951	[Message]	Return type of function "main" must be "int".
E0520953	[Message]	A default template argument cannot be specified on the declaration of a member of a class template outside of its class.
E0520961	[Message]	Use of a type with no linkage to declare a variable with linkage.
E0520962	[Message]	Use of a type with no linkage to declare a function.
E0520965	[Message]	Incorrectly formed universal character name.
E0520966	[Message]	Universal character name specifies an invalid character.
E0520967	[Message]	A universal character name cannot designate a character in the basic character set.
E0520968	[Message]	This universal character is not allowed in an identifier.
E0520969	[Message]	The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros.
E0520976	[Message]	A compound literal is not allowed in an integral constant expression.
E0520977	[Message]	A compound literal of type *NAME* is not allowed.
E0520983	[Message]	typedef name has already been declared (with similar type).
E0520992	[Message]	Invalid macro definition:.
E0520993	[Message]	Subtraction of pointer types "*type1*" and "*type2*" is nonstandard.
E0521012	[Message]	A using-declaration may not name a constructor or destructor.
E0521029	[Message]	Type containing an unknown-size array is not allowed.
E0521030	[Message]	A variable with static storage duration cannot be defined within an inline function.
E0521031	[Message]	An entity with internal linkage cannot be referenced within an inline function with external linkage.
E0521036	[Message]	The reserved identifier "*symbol*" may only be used inside a function.
E0521037	[Message]	This universal character cannot begin an identifierl.
E0521038	[Message]	Expected a string literal.
E0521039	[Message]	Unrecognized STDC pragma.
E0521040	[Message]	Expected "ON", "OFF", or "DEFAULT".
E0521041	[Message]	A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.
E0521045	[Message]	Invalid designator kind.
E0521048	[Message]	Conversion between real and imaginary yields zero.
E0521049	[Message]	An initializer cannot be specified for a flexible array member.
E0521051	[Message]	Standard requires that *NAME* be given a type by a subsequent declaration ("int" assumed).
E0521052	[Message]	A definition is required for inline *NAME*.
E0521055	[Message]	Types cannot be declared in anonymous unions.

E0521056	[Message]	Returning pointer to local variable.
E0521057	[Message]	Returning pointer to local temporary.
E0521062	[Message]	The other declaration is %p.
E0521072	[Message]	A declaration cannot have a label.
E0521139	[Message]	The "template" keyword used for syntactic disambiguation may only be used within a template.
E0521144	[Message]	Storage class must be auto or register.
E0521158	[Message]	void return type cannot be qualified.
E0521203	[Message]	Parameter *parameter* may not be redeclared in a catch clause of function try block.
E0521206	[Message]	"template" must be followed by an identifier.
E0521273	[Message]	Alignment-of operator applied to incomplete type.
E0521313	[Message]	Hexadecimal floating-point constants are not allowed.
E0521319	[Message]	Fixed-point operation result is out of range.
E0521348	[Message]	Declaration hides "*symbol*".
E0521352	[Message]	Expected "SAT" or "DEFAULT".
E0521381	[Message]	Carriage return character in source line outside of comment or character/string literal.
	[Explanation]	Carriage return character (\r) in source line outside of comment or character/string literal.
E0521420	[Message]	Some enumerator values cannot be represented by the integral type underlying the enum type.
E0521537	[Message]	Unrecognized calling convention *xxx* must be one of:
E0521539	[Message]	Option "--uliterals" can be used only when compiling C.
E0521578	[Message]	case label value has already appeared in this switch at line *number*.
E0521582	[Message]	The option to list macro definitions may not be specified when compiling more than one translation unit.
E0521584	[Message]	Parentheses around a string initializer are nonstandard.
E0521603	[Message]	Variable of incomplete type "*variable*" cannot be placed into the section.
E0521604	[Message]	Illegal section attribute.
E0521605	[Message]	Illegal #pragma *character string* syntax.
E0521606	[Message]	"*function*" has already been placed into another section.
	[Explanation]	A "#pragma text" has already been specified for function "*function*". It cannot be put into a different section.
E0521608	[Message]	#pragma asm is not allowed outside of function.
E0521609	[Message]	The #pragma endasm for this #pragma asm is missing.
E0521610	[Message]	The #pragma asm for this #pragma endasm is missing.
E0521612	[Message]	Duplicate interrupt hander for "*request*".
E0521613	[Message]	Interrupt request name "*request*" not supported.
E0521614	[Message]	Duplicate #pragma interrupt for this function.

E0521615	[Message]	Duplicate #pragma smart_correct for this function "*function*".
	[Explanation]	A "#pragma smart_correct" has already been specified for function "*function*".
E0521616	[Message]	*Type* "*symbol*" has already been placed into another section (declared as extern).
E0521617	[Message]	*Type* "*symbol*" has already been placed into another section.
E0521618	[Message]	*Type* "*symbol*" has already been declared with #pragma section.
E0521621	[Message]	Cannot write I/O register "*register name*".
E0521622	[Message]	Cannot read I/O register "*register name*".
E0521623	[Message]	Cannot use *expanded specification*. Device must be specified.
E0521625	[Message]	Cannot set interrupt level for "*request*".
E0521626	[Message]	*Specification character string* is specified for function "*function name*", previously specified #pragma inline is ignored.
E0521627	[Message]	Function for #pragma smart_correct is same.
E0521628	[Message]	Function for #pragma smart_correct "*function*" is undefined.
E0521630	[Message]	Could not close symbol file "*file name*".
E0521633	[Message]	Section name is not specified.
E0521635	[Message]	"*variable name*" has already been placed into "*section name*" section in symbol file. The latter is ignored.
E0521636	[Message]	"*variable name*" has already been placed into "*section name*" section in symbol file. #pragma is ignored.
E0521637	[Message]	Illegal binary digit.
E0521638	[Message]	First argument for *special function name*()must be integer constant.
E0521639	[Message]	Function "*function name*" specified as "direct" can not be allocated in text.
E0521640	[Message]	Function allocated in text can not be specified #pragma interrupt with "direct".
E0521641	[Message]	FE level interrupt not supported.
E0521642	[Message]	Cannnot give a name for "*attribute*" section.
E0521643	[Message]	"direct" cannot be specified for plural interrupt.
E0521644	[Message]	Reduced exception handler option of device is available. Address of the handler-maybe overlaps.
E0521647	[Message]	*character string* is not allowed here.
E0521648	[Message]	Cannot call *type* function "*function name*".
E0521649	[Message]	White space is required between the macro name *NAME* and its replacement text.
E0521650	[Message]	*type* "*symbol name*" has already been declared with other #pragma pic/nopic.
	[Explanation]	There is a "#pragma pin/nopic" specification in conflict with *type* "*symbol name*".
E0523003	[Message]	Expected a section name string.
E0523004	[Message]	expected a section name
	[Explanation]	There is no character string for the section name or an unusable character is used.

E0523005	[Message]	Invalid pragma declaration
	[Explanation]	Write the #pragma syntax in accord with the correct format.
	[Action by User]	The iodefine.h file is generated when a project is created in an integrated development environment. Since the interrupt request names are defined in this file, iodefine.h should be included in a C source file which uses the interrupt request names. The description format of interrupt functions differ in CA78K0R and CC-RL. CC-RL provides the -convert_cc option to aid porting from CA78K0R to CC-RL. Using this option allows some of the descriptions made in the CA78K0R format to be handled by CC-RL.
E0523006	[Message]	"*symbol name*" has already been specified by other pragma
	[Explanation]	Two or more #pragma directives have been specified for one symbol, and such specification is not allowed.
E0523007	[Message]	Pragma may not be specified after definition
	[Explanation]	The #pragma directive precedes definition of the target symbol.
E0523008	[Message]	Invalid kind of pragma is specified to this symbol
	[Explanation]	The given type of #pragma directive is not specifiable for the symbol.
E0523014	[Message]	Invalid binary digit.
E0523018	[Message]	a member qualified with near or far is declared
	[Explanation]	__near or __far cannot be specified for a member when defining a structure or union.
E0523038	[Message]	A struct/union/class has different pack specifications.
E0523044	[Message]	Illegal section naming.
E0523048	[Message]	Illegal reference to interrupt function.
E0523061	[Message]	Argument is incompatible with formal parameter of intrinsic function.
E0523062	[Message]	Return value type does not match the intrinsic function type.
E0523065	[Message]	Cannot assign address constant to initializer for bitfield
	[Action by User]	Do not write an address constant as the initial value of the bit field.
E0523067	[Message]	Type nest is too deep
	[Explanation]	Nesting of the declarator is too deep.
	[Action by User]	Do not write nesting that exceeds the limit of the implementation.
E0523074	[Message]	"*function name*" cannot be used with #pragma rtos_interrupt
	[Explanation]	#pragma rtos_interrupt cannot be specified for "*function name*".
E0523075	[Message]	Combination of address and near/far attribute is incorrect
	[Explanation]	The address specified by #pragma address is a location conflicting with the attributes of __near and __far which are specified as variables.
E0523077	[Message]	Called function should have prototype.
E0523078	[Message]	*xxx* cannot be used in CC-RL.
E0523087	[Message]	Illegal reference to "*function name*"
E0532002	[Message]	Exception *exception* has occurred at compile time.

E0541004	[Message]	Addition/subtraction of __sectop/__secend and a constant are not allowed.
E0541240	[Message]	Illegal naming of section "*section name*".
E0541854	[Message]	Illegal address was specified with #pragma address.
	[Explanation]	The same address is specified for different variables.
E0550200	[Message]	Illegal alignment value.
	[Action by User]	Check the alignment condition specification.
E0550201	[Message]	Illegal character.
	[Action by User]	Check the character.
E0550202	[Message]	Illegal expression.
	[Action by User]	Check the expression.
E0550203	[Message]	Illegal expression (*string*).
	[Action by User]	Check the expression element.
E0550208	[Message]	Illegal expression (labels in different sections).
	[Action by User]	Check the expression.
E0550209	[Message]	Illegal expression (labels must be defined).
	[Action by User]	Check the expression.
E0550212	[Message]	Symbol already defined as *label*.
	[Action by User]	Check the symbol name.
E0550213	[Message]	Label *identifier* redefined.
	[Action by User]	Check the label name.
E0550214	[Message]	*identifier* redefined.
	[Action by User]	Check the label name.
E0550217	[Message]	Illegal operand (cannot use bit I/O register).
	[Action by User]	Check the internal peripheral I/O register.
E0550220	[Message]	Illegal operand (*identifier* is reserved word).
	[Action by User]	Check the operand.
E0550221	[Message]	Illegal operand (label - label).
	[Action by User]	Check the expression.
E0550225	[Message]	Illegal operand (must be evaluated positive or zero).
	[Action by User]	Check the expression.
E0550226	[Message]	Illegal operand (must be even displacement).
	[Action by User]	Check the displacement.
E0550228	[Message]	Illegal operand (must be register).
	[Action by User]	Check the operand.
E0550229	[Message]	Illegal operand (needs base register).
	[Action by User]	Check the operand.

E0550230	[Message]	Illegal operand (range error in displacement).
	[Action by User]	Check the displacement.
E0550231	[Message]	Illegal operand (range error in immediate).
	[Action by User]	Check the immediate.
E0550232	[Message]	Illegal operand (.local parameter).
	[Action by User]	Check the parameter.
E0550234	[Message]	Illegal operand (macro parameter).
	[Action by User]	Check the parameter.
E0550235	[Message]	Illegal operand (macro name).
	[Action by User]	Check "macro name".
E0550236	[Message]	Illegal operand (macro argument).
	[Action by User]	Check the parameter.
E0550237	[Message]	Illegal operand (.irp argument).
	[Action by User]	Check the argument.
E0550238	[Message]	Illegal operand (.irp parameter).
	[Action by User]	Check the parameter.
E0550242	[Message]	Illegal operand (label is already defined on *section*).
	[Action by User]	Check the label.
E0550244	[Message]	Illegal origin value (*value*).
	[Action by User]	Check the value.
E0550245	[Message]	*identifier* is reserved word.
	[Action by User]	Check the code.
E0550246	[Message]	Illegal section.
	[Action by User]	Check the code.
E0550247	[Message]	Illegal size value.
	[Action by User]	Check the specification.
E0550248	[Message]	Illegal symbol reference (*symbol*).
	[Action by User]	Check the symbol.
E0550249	[Message]	Illegal syntax.
	[Action by User]	Check the code.
E0550250	[Message]	Illegal syntax *string*.
	[Action by User]	Check the code.
E0550260	[Message]	Token too long.
	[Explanation]	Token too long. The boundary value is 4,294,967,294.
	[Action by User]	Check the token length.

E0550271	[Message]	"*string1*" conflicts with previously specified "*string2*".
	[Explanation]	"*string1*" conflicts with previously specified "*string2*". Check the description of the source. Note: "align=0" means that align is not specified in the .section directive.
E0550272	[Message]	"*string*" required.
	[Action by User]	Add the specification of "*string*" to the relevant line.
E0550601	[Message]	"*path-name*" specified by the "*character string*" option is a folder. Specify an input file.
E0550602	[Message]	The file "*file-name*" specified by the "*character string*" option is not found.
	[Action by User]	Check if the file exists.
E0550603	[Message]	"*path-name*" specified by the "*character string*" option is a folder. Specify an output file.
E0550604	[Message]	The output folder "*folder-name*" specified by the "*character string*" option is not found.
E0550605	[Message]	"*string2*" specified by the "*string1*" option is a file. Specify a folder.
E0550606	[Message]	The folder "*string2*" specified by the "*string1*" option is not found.
E0550607	[Message]	"*path-name*" specified by the "*character string*" option is not found.
	[Explanation]	"*path-name*" (file or folder name) specified by the "*character string*" option was not found.
E0550608	[Message]	The "*character string*" option is not recognized.
E0550609	[Message]	The "*character string*" option can not have an argument.
E0550610	[Message]	The "*character string*" option requires an argument.
E0550611	[Message]	The "*character string*" option can not have an argument.
E0550612	[Message]	The "*character string*" option requires an argument.
	[Explanation]	The "*character string*" option requires an argument.
	[Action by User]	Specify an argument.
E0550613	[Message]	Invalid argument for the "*character string*" option.
E0550617	[Message]	Invalid argument for the "*character string*" option.
E0550624	[Message]	The "-cpu" option must be specified.
E0550625	[Message]	Cannot find device file.
E0550629	[Message]	Command file "*file-name*" is read more than once.
E0550630	[Message]	Command file "*file-name*" can not be read.
E0550631	[Message]	Syntax error in command file "*file-name*".
E0550632	[Message]	Failed to create temporary folder.
E0550633	[Message]	The argument for the "*string*" option must be a folder when multiple source files are specified.
E0550637	[Message]	Failed to delete a temporary folder "*folder-name*".
E0550638	[Message]	Failed to open an input file "*file-name*".
E0550639	[Message]	Failed to open an output file "*file-name*".

E0550640	[Message]	Failed to close an input file "*file-name*".
E0550641	[Message]	Failed to write an output file "*file-name*".
E0550645	[Message]	"*character string2*" specified in the "*character string1*" option is not available.
E0550647	[Message]	The "*string*" option is specified more than once. The latter is valid.
E0550649	[Message]	The "*string2*" option is ignored when the "*string1*" option and the "*string2*" option are inconsistent.
E0550701	[Message]	Failed to delete a temporary file "*file-name*".
E0551200	[Message]	Syntax error.
	[Explanation]	There is an error in the assembly source code.
	[Action by User]	Check the assembly source code.
E0551202	[Message]	Illegal register.
	[Explanation]	There is a register that cannot be specified as an operand.
	[Action by User]	Check which registers can be specified as operands.
E0551203	[Message]	Relocatable symbol is not allowed.
	[Explanation]	There is a relocatable symbol at a location not allowed.
	[Action by User]	Check the description format of the respective location.
E0551204	[Message]	Illegal operands.
	[Explanation]	An illegal operand is specified.
	[Action by User]	Check the formats that can be specified as operands.
E0551205	[Message]	Illegal string.
	[Explanation]	There is an error in the string.
	[Action by User]	Check if there are errors in the string.
E0551206	[Message]	"$" is not allowed.
	[Explanation]	There is "$" where it is not allowed.
	[Action by User]	Check that there is no "$" where it is not allowed.
E0551207	[Message]	"*string*" is not allowed.
	[Action by User]	Check the description format of the respective location.
E0551208	[Message]	Illegal operation ("op").
	[Explanation]	There is an error in the description of "op" operation.
	[Action by User]	Check the description of "op" operation.
E0551209	[Message]	Illegal 1st operand in bit position specifier.
	[Action by User]	Check the description of the 1st operand of the bit position specifier.
E0551210	[Message]	Byte separation operator for bit reference is not allowed.
	[Action by User]	Apply the separation operator to the 1st operand for bit reference.
E0551211	[Message]	Only bit symbols are allowed.
	[Action by User]	Check that there are only bit symbols.

E0551212	[Message]	Illegal bit position specifier.
	[Action by User]	Check the description of the bit position specifier.
E0551213	[Message]	Operand or right parenthesis is missing.
	[Explanation]	Either a right parenthesis is missing or there is no expression to be targeted by the operator.
	[Action by User]	Check that there is a right parenthesis to match each left parenthesis or there is an expression to be targeted by the operator.
E0551214	[Message]	Illegal operation ("op").
	[Action by User]	Check the format of the op operator.
E0551215	[Message]	Illegal label reference.
	[Action by User]	Check the description of the label.
E0551218	[Message]	Illegal expression (-label).
	[Explanation]	An expression of the (-label) format is not allowed.
	[Action by User]	Check the expression.
E0551219	[Message]	Illegal label reference.
	[Explanation]	Operation or reference of a label is invalid.
	[Action by User]	Check the Operation or reference of a label.
E0551220	[Message]	Undefined symbol is not allowed.
	[Explanation]	There is an undefined symbol where it is not allowed.
	[Action by User]	Check the symbol definition.
E0551221	[Message]	Section name is not allowed.
	[Explanation]	There is a section name where it is not allowed.
	[Action by User]	Check which section names are allowed.
E0551222	[Message]	Illegal character.
	[Explanation]	Failed to read characters.
	[Action by User]	Check the code.
E0551223	[Message]	Closing single quotation mark is missing.
	[Explanation]	A single quotation (') is not closed.
	[Action by User]	Check the single quotation (') is not closed.
E0551224	[Message]	Illegal string.
	[Explanation]	Failed to read strings.
	[Action by User]	Check the code.
E0551225	[Message]	Closing double quotation mark is missing.
	[Explanation]	A double quotation (") is not closed.
	[Action by User]	Check if the double quotation (") is closed.
E0551226	[Message]	Illegal string in expression.
	[Explanation]	There is a string in the middle of an expression.

E0551227	[Message]	'?' is not allowed.
	[Explanation]	'?' is not handled as an alphanumeric character. It cannot be used in a symbol name.
E0551228	[Message]	Numeric description does not match -base_number option.
E0551229	[Message]	Invalid binary number.
	[Action by User]	Check if the binary notation is correct.
E0551230	[Message]	Invalid octal number.
	[Action by User]	Check if the octal notation is correct.
E0551231	[Message]	Invalid decimal number.
	[Action by User]	Check if the decimal notation is correct.
E0551232	[Message]	Invalid hexadecimal number.
	[Action by User]	Check if the hexadecimal notation is correct.
E0551233	[Message]	Too many operands.
	[Action by User]	Specify operands for the correct number.
E0551234	[Message]	Closing bracket is missing.
	[Explanation]	There is no right bracket.
E0551236	[Message]	Illegal tilde operation.
	[Explanation]	There is an error in the description of the tilde.
E0551301	[Message]	Bit number should be in the range 0-7.
E0551302	[Message]	Specified address is out of saddr area.
	[Explanation]	The value specified for the operand is outside of the saddr area.
E0551303	[Message]	Specified address is out of SFR area.
	[Explanation]	The value specified for the operand is outside of the sfr area.
E0551304	[Message]	Specified address is out of callt table area.
	[Explanation]	The value specified for the operand is outside of the callt table area.
E0551305	[Message]	Specified value is out of 8-bit integer.
	[Explanation]	The value specified for the operand exceeds the 8-bit width.
E0551306	[Message]	Specified value is out of 16-bit integer.
	[Explanation]	The value specified for the operand exceeds the 16-bit width.
E0551307	[Message]	Specified value is out of 20-bit integer.
	[Explanation]	The value specified for the operand exceeds the 20-bit width.
E0551308	[Message]	Specified value is out of 32-bit integer.
	[Explanation]	The value specified for the operand exceeds the 32-bit width.
E0551309	[Message]	Odd number is not allowed.
E0551310	[Message]	Only "1" is allowed.
E0551311	[Message]	Specified value is out of range 1-7.
E0551312	[Message]	Specified value is out of range 1-15.

E0551313	[Message]	"*reg*" is not allowed.
	[Explanation]	Register *reg* is not allowed here.
	[Action by User]	Check which operands are allowed.
E0551314	[Message]	Only HL register is allowed.
E0551315	[Message]	Only ES register is allowed.
E0551316	[Message]	Only SFR register is allowed.
	[Explanation]	There is an illegal SFR or a control register.
E0551317	[Message]	Forward reference of SFR is not allowed.
E0551401	[Message]	Illegal operand "*operand*".
	[Explanation]	There is an error in the description of the operand.
	[Action by User]	Check the description of the operand.
E0551402	[Message]	Illegal instruction.
	[Explanation]	The instruction type is illegal.
E0551403	[Message]	Illegal operand of .DB8 directive.
	[Explanation]	A separation operator cannot be set for the operand of the .DB8 pseudo instruction.
	[Action by User]	Check if the .DB8 operand is described correctly.
E0551404	[Message]	Illegal address description of .VECTOR directive.
	[Action by User]	Check if the address of the .VECTOR pseudo instruction is specified correctly.
E0551405	[Message]	Illegal $MIRROR declaration.
	[Explanation]	$MIRROR cannot be specified for the symbol.
	[Action by User]	Check whether the $MIRROR pseudo instruction specifies an external reference name.
E0551406	[Message]	Any symbol name starting with a period must not be used for "*directive*".
E0551501	[Message]	Multiple source files are not allowed when the "-output" option is specified.
E0562000	[Message]	Invalid option : "*option*"
	[Explanation]	*option* is not supported.
E0562001	[Message]	Option "*option*" cannot be specified on command line
	[Explanation]	*option* cannot be specified on the command line.
	[Explanation]	Specify this option in a subcommand file.
E0562002	[Message]	Input option cannot be specified on command line
	[Explanation]	The input option was specified on the command line.
	[Action by User]	Input file specification on the command line should be made without the input option.
E0562003	[Message]	Subcommand option cannot be specified in subcommand file
	[Explanation]	The -subcommand option was specified in a subcommand file. The -subcommand option cannot be nested.

E0562004	[Message]	Option "*option1*" cannot be combined with option "*option2*"
	[Explanation]	*option1* and *option2* cannot be specified simultaneously.
E0562005	[Message]	Option "*option*" cannot be specified while processing "*process*"
	[Explanation]	*option* cannot be specified for *process*.
E0562006	[Message]	Option "*option1*" is ineffective without option "*option2*"
	[Explanation]	*option1* requires *option2* be specified.
E0562010	[Message]	Option "*option*" requires parameter
	[Explanation]	*option* requires a parameter to be specified.
E0562011	[Message]	Invalid parameter specified in option "*option*" : "*parameter*"
	[Explanation]	An invalid parameter was specified for *option*.
E0562012	[Message]	Invalid number specified in option "*option*" : "*value*"
	[Explanation]	An invalid value was specified for *option*.
	[Action by User]	Check the range of valid values.
E0562013	[Message]	Invalid address value specified in option "*option*" : "*address*"
	[Explanation]	The address *address* specified in *option* is invalid.
	[Action by User]	A hexadecimal address between 0 and FFFFFFFF should be specified.
E0562014	[Message]	Illegal symbol/section name specified in "*option*" : "*name*"
	[Explanation]	The section or symbol name specified in *option* uses an illegal character.
E0562016	[Message]	Invalid alignment value specified in option "*option*" : "*alignment value*"
	[Explanation]	The alignment value specified in *option* is invalid.
	[Action by User]	1, 2, 4, 8, 16, or 32 should be specified.
E0562020	[Message]	Duplicate file specified in option "*option*" : "*file*"
	[Explanation]	The same file was specified twice in *option*.
E0562022	[Message]	Address ranges overlap in option "*option*" : "*address range*"
	[Explanation]	Address ranges *address range* specified in *option* overlap.
E0562100	[Message]	Invalid address specified in cpu option : "*address*"
	[Explanation]	An address was specified with the -cpu option that cannot be specified for a cpu.
E0562101	[Message]	Invalid address specified in option "*option*" : "*address*"
	[Explanation]	The *address* specified in *option* exceeds the address range that can be specified by the cpu or the range specified by the cpu option.
E0562110	[Message]	Section size of second parameter in rom option is not 0 : "*section*"
	[Explanation]	The second parameter in the -rom option specifies "*section*" with non-zero size.
E0562111	[Message]	Absolute section cannot be specified in "*option*" option : "*section*"
	[Explanation]	An absolute address section was specified in *option*.
E0562114	[Message]	The generated duplicate section name "*section*" is confused
	[Explanation]	A section with the same name *section* appeared more than once and could not be processed.

E0562120	[Message]	Library "*file*" without module name specified as input file
	[Explanation]	A library file without a module name was specified as the input file.
E0562121	[Message]	Input file is not library file : "*file*(*module*)"
	[Explanation]	The file specified by *file* (*module*) as the input file is not a library file.
E0562130	[Message]	Cannot find file specified in option "*option*" : "*file*"
	[Explanation]	The file specified in *option* could not be found.
E0562131	[Message]	Cannot find module specified in option "*option*" : "*module*"
	[Explanation]	The module specified in *option* could not be found.
E0562132	[Message]	Cannot find "*name*" specified in option "*option*"
	[Explanation]	The symbol or section specified in *option* does not exist.
E0562133	[Message]	Cannot find defined symbol "*name*" in option "*option*"
	[Explanation]	The externally defined symbol specified in *option* does not exist.
E0562134	[Message]	Reserved section name "*section*"
	[Explanation]	" *section*" is the reservation name used by a linker.
	[Action by User]	Check if the section name is correct.
E0562135	[Message]	[V1.06 or earlier] Interrupt number "*vector table address*" has invalid interrupt jump address : "*symbol*" [V1.07 or later] Interrupt table address "*vector table address*" has invalid interrupt jump address : "*symbol*"
	[Explanation]	"*symbol*" cannot be specified as an interrupt function.
	[Action by User]	Check the description of the option and source file.
E0562140	[Message]	Symbol/section "*name*" redefined in option "*option*"
	[Explanation]	The symbol or section specified in *option* has already been defined.
E0562141	[Message]	Module "*module*" redefined in option "*option*"
	[Explanation]	The module specified in *option* has already been defined.
E0562142	[Message]	[V1.06 or earlier] Interrupt number "*vector table address*" of "*section*" has multiple definition [V1.07 or later] Interrupt table address "*vector table address*" of "*section*" has multiple definition
	[Explanation]	Vector number definition was made multiple times in vector table *section*. Only one address can be specified for a vector number.
	[Action by User]	Check and correct the code in the source file.
E0562200	[Message]	Illegal object file : "*file*"
	[Explanation]	A format other than ELF format was input.
	[Action by User]	Since CA78K0R and CC-RL have different object file formats, an object file created in CA78K0R cannot be linked by CC-RL. If an object file of CA78K0R has been used, recreate an object file specifically for CC-RL and link that file.

E0562201	[Message]	Illegal library file : "*file*"
	[Explanation]	*file* is not a library file.
	[Action by User]	Check whether a library file of CA78K0R has been specified. Since CA78K0R and CC-RL have different object file formats, a library file created in CA78K0R cannot be linked by CC-RL. Recreate a library file specifically for CC-RL and link that file.
E0562204	[Message]	Unsupported device file : "*file*"
	[Explanation]	"*file*" cannot be read as the device file.
	[Action by User]	Check the device file.
E0562210	[Message]	Invalid input file type specified for option "*option*" : "*file*(*type*)"
	[Explanation]	When specifying *option*, a *file* (*type*) that cannot be processed was input.
E0562211	[Message]	Invalid input file type specified while processing "*process*" : "*file*(*type*)"
	[Explanation]	A *file* (*type*) that cannot be processed was input during processing *process*.
E0562212	[Message]	"*option*" cannot be specified for inter-module optimization information in "*file*"
	[Explanation]	The option *option* cannot be used because *file* includes inter-module optimization information.
	[Action by User]	Do not specify the goptimize option at compilation or assembly.
E0562220	[Message]	Illegal mode type "*mode type*" in "*file*"
	[Explanation]	A file with a different *mode type* was input.
E0562221	[Message]	Section type mismatch : "*section*"
	[Explanation]	Sections with the same name but different attributes (whether initial values present or not) were input.
E0562224	[Message]	Section type (relocation attribute) mismatch : "*section*"
	[Explanation]	Sections with the same name but different relocation attributes were specified.
E0562225	[Message]	Device file mismatch "*device file*" in "*input file*"
	[Explanation]	An object file created using a different device file is attempted to be linked or the device file used when creating an object file does not match the device file specified by the -device option.
E0562300	[Message]	Duplicate symbol "*symbol*" in "*file*"
	[Explanation]	There are duplicate occurrences of *symbol*.
E0562301	[Message]	Duplicate module "*module*" in "*file*"
	[Explanation]	There are duplicate occurrences of *module*.
E0562310	[Message]	Undefined external symbol "*symbol*" referenced in "*file*"
	[Explanation]	An undefined symbol *symbol* was referenced in *file*.
E0562311	[Message]	Section "*section1*" cannot refer to overlaid section : "*section2-symbol*"
	[Explanation]	A symbol defined in *section1* was referenced in *section2* that is allocated to the same address as *section1* overlaid.
	[Action by User]	*section1* and *section2* must not be allocated to the same address.

E0562320	[Message]	Section address overflowed out of range : "*section*"
	[Explanation]	The address of *section* exceeds the usable address range.
	[Action by User]	Check the allocation address and size of the section.
E0562321	[Message]	Section "*section1*" overlaps section "*section2*"
	[Explanation]	The addresses of *section1* and *section2* overlap.
	[Action by User]	Change the address specified by the start option.
E0562325	[Message]	Section "*section*" steps over the border of "*border*"
	[Explanation]	*section* is allocated to extend across *border*.
E0562330	[Message]	Relocation size overflow : "*file*"-"*section*"-"*offset*
	[Explanation]	The result of the relocation operation exceeded the relocation size. Possible causes include inaccessibility of a branch destination, and referencing of a symbol which must be located at a specific address.
	[Action by User]	Ensure that the referenced symbol at the offset position of section in the source list is placed at the correct position.
E0562332	[Message]	Relocation value is odd number : "*file*"-"*section*"-"*offset*"
	[Explanation]	The result of the relocation operation is an odd number.
	[Action by User]	Check for problems in calculation of the position at *offset* in *section* in the source list.
E0562340	[Message]	Symbol name "*file*"-"*section*"-"*symbol*" is too long
	[Explanation]	The length of "*symbol*" in "*section*" exceeds the assembler translation limit.
	[Action by User]	To output a symbol address file, use a symbol name that is no longer than the assembler translation limit.
E0562350	[Message]	Section "*section*" cannot be placed on the "*area*".
	[Explanation]	When the -self option is specified, a "*section*" cannot be allocated to "*area*".
E0562351	[Message]	Section "*section*" cannot be placed on the "*area*".
	[Explanation]	When the -ocdtr option is specified, a "*section*" cannot be allocated to "*area*".
E0562352	[Message]	Section "*section*" cannot be placed on the "*area*".
	[Explanation]	When the -ocdhpi option is specified, a "*section*" cannot be allocated to "*area*".
E0562353	[Message]	Section "*section*" address overflowed out of range "*area*".
	[Explanation]	When the -selfw option is specified, a "*section*" cannot be allocated to extend across "*area*".
E0562354	[Message]	Section "*section*" address overflowed out of range "*area*".
	[Explanation]	When the -ocdtrw option is specified, a "*section*" cannot be allocated to extend across "*area*".
E0562355	[Message]	Section "*section*" address overflowed out of range "*area*".
	[Explanation]	When the -ocdhpiw option is specified, a "*section*" cannot be allocated to extend across "*area*".
E0562360	[Message]	CRC result cannot be placed on the "*area*".
	[Explanation]	When the -self option is specified, the CRC result cannot be placed in "*area*".

E0562361	[Message]	CRC result cannot be placed on the "*area*".
	[Explanation]	When the -ocdtr option is specified, the CRC result cannot be placed in "*area*".
E0562362	[Message]	CRC result cannot be placed on the "*area*".
	[Explanation]	When the -ocdhpi option is specified, the CRC result cannot be placed in "*area*".
E0562363	[Message]	CRC result address overflowed out of range "*area*".
	[Explanation]	When the -selfw option is specified, the CRC result cannot be placed to extend across "*area*".
E0562364	[Message]	CRC result address overflowed out of range "*area*".
	[Explanation]	When the -ocdtrw option is specified, the CRC result cannot be placed to extend across "*area*".
E0562365	[Message]	CRC result address overflowed out of range "*area*".
	[Explanation]	When the -ocdhpiw option is specified, the CRC result cannot be placed to extend across "*area*".
E0562410	[Message]	Address value specified by map file differs from one after linkage as to "*symbol*"
	[Explanation]	The address of *symbol* differs between the address within the external symbol allocation information file used at compilation and the address after linkage.
	[Action by User]	Check (1) to (3) below. (1) Do not change the program before or after the map option specification at compilation. (2) rlink optimization may cause the sequence of the symbols after the map option specification at compilation to differ from that before the map option. Disable the map option at compilation or disable the rlink option for optimization. (3) When the tbr option or #pragma tbr is used, optimization by the compiler may delete symbols after the map option specification at compilation. Disable the map option at compilation or disable the tbr option or #pragma tbr.
E0562411	[Message]	Map file in "*file*" conflicts with that in another file
	[Explanation]	Different external symbol allocation information files were used by the input files at compilation.
E0562412	[Message]	Cannot open file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be opened.
	[Action by User]	Check whether the file name and access rights are correct.
E0562413	[Message]	Cannot close file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be closed. There may be insufficient disk space.
E0562414	[Message]	Cannot read file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be read. There may be insufficient disk space.
E0562415	[Message]	Illegal map file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) has an illegal format.
	[Action by User]	Check whether the file name is correct.

E0562416	[Message]	Order of functions specified by map file differs from one after linkage as to "*function name*"
	[Explanation]	The sequences of a function *function name* and those of other functions are different between the information within the external symbol allocation information file used at compilation and the location after linkage. The address of static within the function may be different between the external symbol allocation information file and the result after linkage.
E0562417	[Message]	Map file is not the newest version : "*file name*"
	[Explanation]	The external symbol allocation information file is not the latest version.
E0562420	[Message]	"*file1*" overlap address "*file2*" : "*address*"
	[Explanation]	The address specified for *file1* is the same as that specified for *file2*.
E0562600	[Message]	Library "*library*" requires "*licence edition*"
	[Explanation]	The "*library*" requires the "*edition*" edition.
E0580001	[Message]	"SMSG%s" cannot be specified as dst.
	[Explanation]	Values cannot be written to SMSG0 or SMSG15.
	[Action by User]	Specify an operand other than SMSG0 or SMSG15.
E0580002	[Message]	"%s" is out of range (%d-%d).
	[Explanation]	The value of the operand is outside the specifiable range.
	[Action by User]	Check the value of the operand.
E0580003	[Message]	label cannot be specified as memory operand.
	[Explanation]	A label cannot be written in the offset of a memory operand.
	[Action by User]	Check the code for the operand.
E0580004	[Message]	branch destination is out of program area.
	[Explanation]	The branch destination of a branch instruction is outside the program.
	[Action by User]	Check the branch distance from the branch instruction.
E0580005	[Message]	invalid address format.
	[Action by User]	Specify only a numeric value or label for $addr.
E0580006	[Message]	unknown label "%s".
	[Explanation]	The branch destination label of the branch instruction cannot be found.
	[Action by User]	Check the code for the operand.
E0580007	[Message]	expecting "%s".
	[Explanation]	There is an error in the syntax.
	[Action by User]	Check the code.
E0580008	[Message]	unexpected "%s".
	[Explanation]	There is an error in the syntax.
	[Action by User]	Check the code.

E0580009	[Message]	"%s" is already defined.
	[Explanation]	The label is defined multiple times.
	[Action by User]	Check the label name.
E0580010	[Message]	illegal operation ("%s").
	[Explanation]	There is an error in the expression.
	[Action by User]	Check the code.
E0580011	[Message]	machine instructions cannot exceed 32 instructions.
	[Explanation]	The program has more than 32 instructions.
	[Action by User]	Reduce the program's size.
E0580012	[Message]	illegal symbol ("%s").
	[Explanation]	An illegal character was used in the symbol name.
	[Action by User]	Check the symbol name.
E0580017	[Message]	suffix and prefix cannot be specified together.
	[Explanation]	There is an error in the hexadecimal notation.
	[Action by User]	Only one of prefix and suffix in hexadecimal can be specified.
E0580100	[Message]	specify .SECTION directive.
	[Action by User]	Write the .SECTION directive at the beginning of the program.
E0580104	[Message]	"%s" is already defined.
	[Explanation]	A macro with the same name has already been defined.
	[Action by User]	Check the macro name.
E0580105	[Message]	illegal macro argument.
	[Explanation]	There is an error in the syntax.
	[Action by User]	Check the code.
E0580106	[Message]	actual argument of macro is not matched.
	[Explanation]	The number of actual parameters in a macro reference does not match the number of formal parameters.
	[Action by User]	Check the code.
E0580200	[Message]	illegal syntax.
	[Explanation]	There is an error in the syntax.
	[Action by User]	Check the code.
E0580201	[Message]	include file cannot nest over 4294967294 times.
	[Explanation]	The level of nesting in the $include control directives has exceeded the limit.
E0580202	[Message]	condition assembly directive cannot nest over 4294967294 times.
	[Explanation]	The level of nesting in the $ifdef control directives has exceeded the limit.
E0580203	[Message]	unexpected directive "%s".
	[Explanation]	There is no $if or $ifdef control directive corresponding to "%s".

E0580204	[Message]	expecting directive "%s".
	[Explanation]	There is no $endif control directive corresponding to "%s".
E0580300	[Message]	"%s" is unrecognized.
	[Explanation]	An illegal option name was present.
	[Action by User]	Check the option specification.
E0580301	[Message]	specify the "-o" option.
	[Action by User]	Specify the -o option.
E0580302	[Message]	"%s" option requires an argument.
	[Action by User]	Specify a parameter for the option.
E0580303	[Message]	invalid argument for the "%s" option.
	[Explanation]	The parameter specification in the option is illegal.
	[Action by User]	Check the option specification.
E0580305	[Message]	"%s" specified in the "%s" option is invalid name.
	[Explanation]	The symbol name specified for a -D or -U option is illegal.
	[Action by User]	Check the specification of the -D or -U option.
E0580307	[Message]	cannot open subcommand file "%s".
	[Explanation]	Opening the subcommand file was not possible.
	[Action by User]	Check the state of the subcommand file.
E0580308	[Message]	cannot read subcommand file "%s".
	[Explanation]	The subcommand file was not readable.
	[Action by User]	Check the state of the subcommand file.
E0580309	[Message]	subcommand file "%s" is read more than once.
	[Explanation]	A subcommand file with the same name was specified more than once.
	[Action by User]	Check the subcommand file specifications.
E0580310	[Message]	missing double quotation.
	[Explanation]	Double-quotation marks were missing.
	[Action by User]	Check the code.
E0580311	[Message]	specify the input file.
	[Action by User]	Specify an input file.
E0580312	[Message]	too many input files.
	[Action by User]	Only one input file specification should be made.
E0580313	[Message]	the file "%s" specified by the "%s" option is a folder.
	[Explanation]	The specified output file is a folder.
	[Action by User]	Check the option specification for the output file.

E0580314	[Message]	the output folder "%s" specified by the "%s" option is not found.
	[Explanation]	The output folder was not found.
	[Action by User]	Check the option specification for the output file.
E0580315	[Message]	cannot open input file "%s".
	[Explanation]	Opening the input file was not possible.
	[Action by User]	Check the state of the input file.
E0580316	[Message]	cannot read input file "%s".
	[Explanation]	The input file was not readable.
	[Action by User]	Check the state of the input file.
E0580317	[Message]	cannot open output file "%s".
	[Explanation]	Opening the output file was not possible.
	[Action by User]	Check the state of the output file.
E0580318	[Message]	cannot write output file "%s".
	[Explanation]	Attempted writing to the output file failed.
	[Action by User]	Check the state of the output file.

10.5.3 Fatal errors

Table 10.4 Fatal Errors

F0520003	[Message]	#include file "*file*" includes itself.
	[Explanation]	#include file "*file*" includes itself. Correct the error.
F0520004	[Message]	Out of memory.
	[Action by User]	Out of memory. Close other applications, and perform the compile again.
F0520005	[Message]	Could not open source file "*file*".
F0520013	[Message]	Expected a file name.
F0520035	[Message]	#error directive: *character string*
	[Explanation]	There is an "#error" directive in the source file.
F0520143	[Message]	Program too large or complicated to compile.
F0520163	[Message]	Could not open temporary file *xxx*.
F0520164	[Message]	Name of directory for temporary files is too long (*xxx*).
F0520182	[Message]	Could not open source file *xxx* (no directories in search list).
F0520189	[Message]	Error while writing "*file*" file.
F0520563	[Message]	Invalid preprocessor output file.
F0520564	[Message]	Cannot open preprocessor output file.
F0520571	[Message]	Invalid option: *option*
F0520642	[Message]	Cannot build temporary file name.
F0520920	[Message]	Cannot open output file: *xxx*
F0523029	[Message]	Cannot open rule file
	[Explanation]	The file specified in the -misra2004="*file name*" or -misra2012="*file name*" option cannot be opened.
F0523030	[Message]	Incorrect description "*file name*" in rule file
	[Explanation]	The file specified in the -misra2004="*file name*" or -misra2012="*file name*" option includes illegal code.
F0523031	[Message]	Rule "*rule number*" is unsupported
	[Explanation]	The number of a rule that is not supported was specified.
F0523061	[Message]	argument is incompatible with formal parameter of intrinsic function
F0523062	[Message]	return value type does not match the intrinsic function
F0523088	[Message]	Bit position is out of range.
F0530320	[Message]	Duplicate symbol "*symbol name*".
F0530800	[Message]	Type of symbol "*symbol-name*" differs between files.
F0530808	[Message]	Alignment of variable "*variable-name*" differs between files.
F0530810	[Message]	#pragma directive for symbol "*symbol-name*" differs between files.
F0531003	[Message]	The function "*function*" specified by the "*option*" option is not exist.

F0533015	[Message]	Symbol table overflow.
	[Explanation]	The number of symbols generated by the compiler exceeded the limit.
F0533021	[Message]	Out of memory.
	[Explanation]	Memory is insufficient.
	[Action by User]	Close other applications and recompile the program.
F0533300	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.
F0533301	[Message]	Cannot close an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be closed.
F0533302	[Message]	Cannot read an intermediate file.
	[Explanation]	An error occurred during reading of a temporary file.
F0533303	[Message]	Cannot write to an intermediate file.
	[Explanation]	An error occurred during writing of a temporary file.
F0533306	[Message]	Compilation was interrupted.
	[Explanation]	During compilation, an interrupt due to entry of the Cntl + C key combination was detected.
F0533330	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.
F0540027	[Message]	Cannot read file "*file-name*".
F0540204	[Message]	Illegal stack access.
	[Explanation]	Attempted usage of the stack by a function has exceeded 64K bytes.
F0540300	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.
F0540301	[Message]	Cannot close an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be closed.
F0540302	[Message]	Cannot read an intermediate file.
	[Explanation]	An error occurred during reading of a temporary file.
F0540303	[Message]	Cannot write to an intermediate file.
	[Explanation]	An error occurred during writing of a temporary file.
F0540400	[Message]	Different parameters are set for the same #pramga "*identifier*".
F0550503	[Message]	Cannot open file *file*.
	[Action by User]	Check the file.
F0550504	[Message]	Illegal section kind.
	[Action by User]	Check the section type specification.
F0550505	[Message]	Memory allocation fault.
	[Action by User]	Check free memory.

F0550506	[Message]	Memory allocation fault (*string*).
	[Action by User]	Check free memory.
F0550507	[Message]	Overflow error (*string*).
	[Explanation]	Ran out of working space while processing the expression. Change it to a simpler expression.
	[Action by User]	Check the expression.
F0550508	[Message]	*identifier* undefined.
	[Action by User]	Check the identifier.
F0550509	[Message]	Illegal pseudo(*string*) found.
	[Action by User]	Check the directive.
F0550510	[Message]	*string* unexpected.
	[Action by User]	Check the directive.
F0550511	[Message]	*string* unmatched.
	[Action by User]	Check the conditional assembly control instruction.
F0550512	[Message]	$if, $ifn, etc. too deeply nested.
	[Explanation]	4294967294 or more levels of nesting have been used in the conditional assembly control instruction.
	[Action by User]	Check the nesting.
F0550513	[Message]	Unexpected EOF in *string*.
	[Explanation]	There is no .endm directive corresponding to *string* directive.
	[Action by User]	Check the directive.
F0550514	[Message]	Argument table overflow.
	[Explanation]	4294967294 or more actual parameters have been used.
	[Action by User]	Check the actual arguments.
F0550516	[Message]	Local symbol value overflow.
	[Explanation]	The number of symbols generated automatically via the .local directive exceeds the maximum limit (4294967294).
	[Action by User]	Check the directive.
F0550526	[Message]	Devicefile version mismatch, cannot use version *version*.
	[Action by User]	Check the device file.
F0550531	[Message]	Too many symbols.
	[Explanation]	The maximum number of symbols that can be included in a single file has been exceeded. The maximum number of symbols that can be included is 4294967294, including symbols registered internally by the assembler.
F0550532	[Message]	Illegal object file (*string*).
	[Explanation]	A file system-dependent error occurred while generating a linkable object file.
	[Action by User]	Check the file system.

F0550534	[Message]	Too many instructions of one file.
	[Explanation]	The maximum number of instructions for one file has been exceeded. The maximum is 10,000,000.
	[Action by User]	Check the number of instructions.
F0550537	[Message]	Section(*section*) address overflowed out of range.
	[Explanation]	The address of the absolute address section is beyond 0xffffffff.
	[Action by User]	When you use .org to specify an absolute address for a section, the final instruction within the section must be allocated to an address up to 0xffffffff.
F0550538	[Message]	Section(*section1*) overlaps section(*section2*).
	[Explanation]	The address range allocated to an absolute address section overlaps with the address range allocated to another section.
	[Action by User]	Check the address specified with .org.
F0550539	[Message]	Relocation entry overflow.
	[Explanation]	There are 16777216 or more symbols that have been registered and referenced.
	[Action by User]	Check the number of symbols.
F0550540	[Message]	Cannot read file *file*.
	[Explanation]	Illegal file, or file size is too long.
	[Action by User]	Check the file.
F0551601	[Message]	Illegal device information specified by "*source*".
F0551604	[Message]	-mirror_source=1 option is not allowed for RL78-S1 core.
F0551605	[Message]	-mirror_region option is not allowed when -dev option is specified.
F0551606	[Message]	-mirror_region option is not allowed when -mirror_source=common option is specified.
F0551607	[Message]	Invalid value is specified as MIRROR area.
F0551608	[Message]	Specify addresses.
F0551609	[Message]	Unreasonable include file nesting.
	[Explanation]	The nesting level of the include is too deep or the function is recursively including itself.
	[Action by User]	Review the include file.
F0551610	[Message]	Unreasonable macro nesting.
	[Explanation]	The nesting level of the macro call is too deep or the function is recursively calling itself.
	[Action by User]	Review the macro definition.
F0563000	[Message]	No input file
	[Explanation]	There is no input file.
F0563001	[Message]	No module in library
	[Explanation]	There are no modules in the library.
F0563002	[Message]	Option "*option1*" is ineffective without option "*option2*"
	[Explanation]	The option *option1* requires that the option *option2* be specified.

F0563003	[Message]	Illegal file format "*file*"
	[Explanation]	*file* has a file format that cannot be used.
F0563004	[Message]	Invalid inter-module optimization information type in "*file*"
	[Explanation]	The "*file*" contains an unsupported inter-module optimization information type.
	[Action by User]	Check if the compiler and assembler versions are correct.
F0563006	[Message]	Option "*option*" cannot be combined with library
	[Explanation]	Option "*option*" cannot be specified together with the library created through the compiler. Check that the correct library file and option are specified.
F0563010	[Message]	No mirror region information
	[Explanation]	The allocation address information of the mirror region is not specified.
	[Action by User]	Check if the -far_rom, -mirror_region, or -dev option is correct
F0563020	[Message]	No cpu information in input files
	[Explanation]	The CPU type cannot be identified from the input file.
	[Action by User]	Check that the binary file is specified with the -binary option and the .obj or .rel files to be linked together exist.
F0563100	[Message]	Section address overflow out of range : "*section*"
	[Explanation]	The address of *section* exceeded the area available.
	[Action by User]	Change the address specified by the start option. For details of the address space, see the user's manual of the device.
F0563102	[Message]	Section contents overlap in absolute section "*section*" in "*file*"
	[Explanation]	Data addresses overlap within an absolute address section.
	[Action by User]	Modify the source program.
F0563103	[Message]	Section size overflow : "*section*"
	[Explanation]	Section "*section*" has exceeded the usable size.
F0563110	[Message]	Illegal cpu type "*cpu type*" in "*file*"
	[Explanation]	A file with a different cpu type was input.
F0563111	[Message]	Illegal encode type "*endian type*" in "*file*"
	[Explanation]	A file with a different endian type was input.
F0563112	[Message]	Invalid relocation type in "*file*"
	[Explanation]	There is an unsupported relocation type in *file*.
	[Action by User]	Ensure the compiler and assembler versions are correct.

F0563113	[Message]	Illegal mode type "*mode*" in "*file*"
	[Explanation]	A "*mode*" file that cannot be mixed is input.
	[Action by User]	Check if the compiler and assembler options and the device file are correct. When you use CS+ to create a project for the RL78-S2 core microcontroller, the project is generated based on the assumption that the division/multiplication and multiply-accumulate unit is to be used. When creating a project with CS+ for a microcontroller that does not have the division/multiplication and multiply-accumulate unit, open the [Common Options] tabbed page on the Property panel of the build tool and select "Not use(-use_mda=not_use)" for the [Use arithmetic unit] setting.
F0563114	[Message]	Illegal cpu type "*CPU type*" in device file "*file*"
	[Explanation]	"*CPU type*" is different.
	[Action by User]	Check if the device file is correct.
F0563115	[Message]	Cpu type in "*file*" is not supported
	[Explanation]	The CPU type specified in "*file*" is not supported. Check if the input file is correct.
F0563121	[Message]	Illegal type of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563122	[Message]	Illegal attribute of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563123	[Message]	Gap is within the limits of the section : "*section*"
	[Explanation]	"*section*" cannot be allocated.
	[Action by User]	Check the source file.
F0563124	[Message]	Illegal alignment of the section : "*section*" in "*file*"
	[Explanation]	"*section*" cannot be allocated by section alignment.
	[Action by User]	Check the source file.
F0563125	[Message]	Illegal kind of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563130	[Message]	Range "*range*" in "*file*" conflicts with that in another file
	[Explanation]	A file which has a different memory area for "*range*" is input.
	[Action by User]	Check if the compiler and assembler options are correct.
F0563140	[Message]	No "*area*" area information in input device file
	[Explanation]	The device file does not contain the "*area*" information.
	[Action by User]	Check if the device file is correct.
F0563150	[Message]	Multiple files cannot be specified while processing "*process*"
	[Explanation]	Multiple files cannot be specified for the *process* processing.
	[Action by User]	Check the file specifications.

F0563200	[Message]	Too many sections
	[Explanation]	The number of sections exceeded the translation limit. It may be possible to eliminate this problem by specifying multiple file output.
F0563201	[Message]	Too many symbols
	[Explanation]	The number of symbols exceeded the translation limit. It may be possible to eliminate this problem by specifying multiple file output.
F0563202	[Message]	Too many modules
	[Explanation]	The number of modules exceeded the translation limit.
	[Action by User]	Divide the library.
F0563203	[Message]	Reserved module name "rlink_generates"
	[Explanation]	rlink_generates_** (** is a value from 01 to 99) is a reserved name used by the optimizing linkage editor. It is used as an .obj or .rel file name or a module name within a library.
	[Action by User]	Modify the name if it is used as a file name or a module name within a library.
F0563204	[Message]	Reserved section name "$sss_fetch"
	[Explanation]	sss_fetch** (sss is any string, and ** is a value from 01 to 99) is a reserved name used by the optimizing linkage editor.
	[Action by User]	Change the symbol name or section name.
F0563300	[Message]	Cannot open file : "*file*"
	[Explanation]	*file* cannot be opened.
	[Action by User]	Check whether the file name and access rights are correct.
F0563301	[Message]	Cannot close file : "*file*"
	[Explanation]	*file* cannot be closed. There may be insufficient disk space.
F0563302	[Message]	Cannot write file : "*file*"
	[Explanation]	Writing to *file* is not possible. There may be insufficient disk space.
F0563303	[Message]	Cannot read file : "*file*"
	[Explanation]	*file* cannot be read. An empty file may have been input, or there may be insufficient disk space.
F0563310	[Message]	Cannot open temporary file
	[Explanation]	A temporary file cannot be opened.
	[Action by User]	Check to ensure the HLNK_TMP specification is correct, or there may be insufficient disk space.
F0563314	[Message]	Cannot delete temporary file
	[Explanation]	A temporary file cannot be deleted. There may be insufficient disk space.
F0563320	[Message]	Memory overflow
	[Explanation]	There is no more space in the usable memory within the optimizing linker.
	[Action by User]	Increase the amount of memory available.
F0563410	[Message]	Interrupt by user
	[Explanation]	An interrupt generated by (Ctrl) + C keys from a standard input terminal was detected.

F0563430	[Message]	The total section size exceeded the limit of the evaluation version of *version*. Please consider purchasing the product. [V1.11 or earlier]
F0563431	[Message]	Incorrect device type, object file mismatch.
	[Explanation]	An unsupported CPU type was input.
	[Action by User]	Check the execution file of the linker and the file specified by the option.
F0563600	[Message]	Option "*option*" requires parameter
	[Explanation]	Parameters have to be specified in *option*.
F0563601	[Message]	Invalid parameter specified in option "*option*" : "*parameter*"
	[Explanation]	An invalid parameter was specified in *option*.
F0563602	[Message]	"*character string*" option requires "*edition*".
	[Explanation]	The "*character string*" option requires the *edition* parameter.
F0580101	[Message]	section name is not specified in the .SECTION directive.
	[Action by User]	Specify a section name in the .SECTION directive.
F0580102	[Message]	"%s" is already specified.
	[Explanation]	A section with the same name has already been defined.
	[Action by User]	Check the section name.
F0580103	[Message]	cannot specify a section name in the .PSECTION directive.
	[Action by User]	Do not write a section name in the .PSECTION directive.
F0580399	[Message]	too many errors.
	[Explanation]	Processing was aborted because there were too many errors.
F0593000	[Message]	‘-cpu’ option is specified twice
	[Explanation]	The -cpu option is specified more than once. Make sure that the specification is valid.
F0593021	[Message]	Memory overflow
	[Explanation]	Memory is insufficient. Close other applications, and generate the library again.
F0593300	[Message]	Cannot open internal file
	[Explanation]	The internal file cannot be opened.
F0593302	[Message]	Cannot input internal file
	[Explanation]	An attempt to read the internal file failed.
F0593303	[Message]	Cannot output internal file
	[Explanation]	An attempt to write to the internal file failed.
F0593305	[Message]	Invalid command parameter "*option-name*"
	[Explanation]	Invalid specification of "*option-name*"
F0593320	[Message]	Command parameter buffer overflow
	[Explanation]	Internal buffer is insufficient.
F0593321	[Message]	Illegal environment variable
	[Explanation]	Environment settings are specified incorrectly. Review the settings.

F0593322	[Message]	Lacking cpu specification
	[Explanation]	The -cpu option is not specified. Check the setting.
F0593324	[Message]	Cannot open subcommand file "*subcommand-file-name*"
	[Explanation]	"*subcommand-file-name*" cannot be opened.
F0593325	[Message]	Cannot close subcommand file
	[Explanation]	The subcommand file cannot be closed.
F0593326	[Message]	Cannot input subcommand file
	[Explanation]	An attempt to read the subcommand file failed.
F0593327	[Message]	Cannot get compiler version
	[Explanation]	The compiler version cannot be acquired.
F0593328	[Message]	Cannot find archive file
	[Explanation]	The component file for the CC-RL is not found. Reinstall the CC-RL.
F0593329	[Message]	Cannot find compiler program
	[Explanation]	The component file for the CC-RL is not found. Reinstall the CC-RL.
F0593330	[Message]	The "*option-name-1*" option and the "*option-name-2*" option are inconsistent
	[Explanation]	The specifications of "*option-name-1*" and "*option-name-2*" are inconsistent. Check the specifications.

10.5.4 Information

Table 10.5 Informations

M0523028	[Message]	Rule *rule number* : *description*
	[Explanation]	Violation of a MISRA-C:2004 rule (indicated by the *rule number* and *description*) was detected.
M0523086	[Message]	Rule *rule number* : *description*
	[Explanation]	Violation of a MISRA-C:2012 rule (indicated by the *rule number* and *description*) was detected.
M0560004	[Message]	"*file*"-"*symbol*" deleted by optimization
	[Explanation]	As a result of symbol_delete optimization, the symbol named *symbol* in *file* was deleted.
M0560005	[Message]	The offset value from the symbol location has been changed by optimization "*file*"-"*section*"-"*symbol* ± offset"
	[Explanation]	As a result of the size being changed by optimization within the range of symbol ± offset, the offset value was changed. Check that this does not cause a problem. To disable changing of the offset value, cancel the specification of the goptimize option on assembly of file.
M0560100	[Message]	No inter-module optimization information in "*file*"
	[Explanation]	No inter-module optimization information was found in *file*. Inter-module optimization is not performed on *file*. To perform inter-module optimization, specify the goptimize option on compiling and assembly.
M0560101	[Message]	No stack information in "*file*"
	[Explanation]	No stack information was found in *file*. *file* may be an assembler output file. The contents of the file will not be in the stack information file output by the optimizing linker.
M0560400	[Message]	Unused symbol "*file*"-"*symbol*"
	[Explanation]	The symbol named *symbol* in *file* is not used.
M0560500	[Message]	Generated CRC code at "*address*"
	[Explanation]	CRC code was generated at *address*.
M0560700	[Message]	Section address overflow out of range : "*section*"
	[Explanation]	The address of "*section*" is beyond the allowable address range.

10.5.5 Warnings

Table 10.6 Warnings

W0511105	[Message]	"*path*" specified by the "*character string*" option is a file. Specify a folder.
W0511106	[Message]	The folder "*folder*" specified by the "*character string*" option is not found.
W0511123	[Message]	The "*character string2*" option is ignored when the "*character string1*" option is specified at the same time.
W0511146	[Message]	"*symbol name*" specified in the "*character string*" option is not allowed for a preprocessor macro.
W0511147	[Message]	The "*character string*" option is specified more than once. The latter is valid.
W0511149	[Message]	The "*character string2*" option is ignored when the "*character string1*" option and the "*character string2*" option are inconsistent.
W0511151	[Message]	The "*character string2*" option is ignored when the "*character string1*" option is not specified.
W0511153	[Message]	Optimization itemoptions were cleared when "-O*character string*" option is specified. Optimization itemoptions need to specify after "-O*character string*" option.
W0511164	[Message]	Duplicate file name. "*file-name*".
W0511180	[Message]	The evaluation period of *version* has expired.
W0511181	[Message]	Error in the Internal information in the file.(*information*)
W0511183	[Message]	License manager is not installed.
	[Action by User]	The license manager is not installed. Install the correct license manager.
W0511184	[Message]	The "-g" option is effective because the "*string*" option is specified.
	[Action by User]	Explicitly specify the "-g" option to suppress output of this message.
W0511185	[Message]	The trial period for the features of the Professional edition expires in *number* days. Please consider purchasing the product of Professional edition.
W0511186	[Message]	The evaluation period for the option "*character string*" of "*version*" is valid for the remaining "*N*" days. After that, it will be implicitly changed to "-Olite". Please consider purchasing the product to continue using "*character string*".
W0511187	[Message]	The evaluation period for the option "*character string*" of "*version*" has expired. It is implicitly changed to "-Olite". Please consider purchasing the product to continue using "*character string*". By explicitly specifying "-Olite" for "-Onothing", this warning message disappears.
W0520009	[Message]	Nested comment is not allowed.
	[Action by User]	Eliminate nesting.
W0520011	[Message]	Unrecognized preprocessing directive.
W0520012	[Message]	Parsing restarts here after previous syntax error.
W0520021	[Message]	Type qualifiers are meaningless in this declaration.
	[Explanation]	Type qualifiers are meaningless in this declaration. Ignored.
W0520026	[Message]	Too many characters in character constant.
	[Explanation]	Too many characters in character constant. Character constants cannot contain more than one character.
W0520027	[Message]	Character value is out of range.

W0520038	[Message]	Directive is not allowed -- an #else has already appeared.
	[Explanation]	Since there is a preceding #else, this directive is illegal.
W0520039	[Message]	Division by zero.
W0520042	[Message]	Operand types are incompatible ("*type1*" and "*type2*").
W0520055	[Message]	Too many arguments in macro invocation.
W0520061	[Message]	Integer operation result is out of range.
W0520062	[Message]	Shift count is negative.
	[Explanation]	Shift count is negative. The behavior will be undefined.
W0520063	[Message]	Shift count is too large.
W0520064	[Message]	Declaration does not declare anything.
W0520068	[Message]	Integer conversion resulted in a change of sign.
W0520069	[Message]	Integer conversion resulted in truncation.
W0520070	[Message]	Incomplete type is not allowed.
W0520076	[Message]	Argument to macro is empty.
W0520077	[Message]	This declaration has no storage class or type specifier.
W0520082	[Message]	Storage class is not first.
	[Explanation]	Storage class is not first. Specify the declaration of the storage class first.
W0520083	[Message]	Type qualifier specified more than once.
W0520099	[Message]	A declaration here must declare a parameter.
W0520108	[Message]	Signed bit field of length 1.
W0520111	[Message]	Statement is unreachable.
W0520117	[Message]	Non-void "*function name*" should return a value.
W0520127	[Message]	Expected a statement.
W0520128	[Message]	Loop is not reachable from preceding code.
W0520138	[Message]	Taking the address of a register variable is not allowed.
W0520140	[Message]	Too many arguments in function call.
W0520152	[Message]	Conversion of nonzero integer to pointer.
W0520159	[Message]	Declaration is incompatible with previous "*declaration*" (declared at line *number*).
W0520161	[Message]	Unrecognized #pragma.
W0520165	[Message]	Too few arguments in function call.
W0520167	[Message]	Argument of type "*type1*" is incompatible with parameter of type "*type2*".
W0520170	[Message]	Pointer points outside of underlying object.
W0520171	[Message]	Invalid type conversion
	[Explanation]	Invalid type conversion
W0520172	[Message]	External/internal linkage conflict with previous declaration.
W0520173	[Message]	Floating-point value does not fit in required integral type.

W0520174	[Message]	Expression has no effect.
	[Explanation]	Expression has no effect. It is invalid.
W0520175	[Message]	Subscript out of range.
W0520177	[Message]	*Type* "*symbol*" was declared but never referenced.
W0520179	[Message]	Right operand of "%" is zero.
W0520180	[Message]	Argument is incompatible with formal parameter.
W0520186	[Message]	Pointless comparison of unsigned integer with zero.
W0520187	[Message]	Use of "=" where "==" may have been intended.
W0520188	[Message]	Enumerated type mixed with another type.
W0520191	[Message]	Type qualifier is meaningless on cast type.
W0520192	[Message]	Unrecognized character escape sequence.
W0520221	[Message]	Floating-point value does not fit in required floating-point type.
W0520222	[Message]	Floating-point operation result is out of range.
W0520223	[Message]	Function *xxx* declared implicitly.
W0520229	[Message]	Bit field cannot contain all values of the enumerated type.
W0520231	[Message]	Declaration is not visible outside of function.
W0520236	[Message]	Controlling expression is constant.
W0520240	[Message]	Duplicate specifier in declaration.
W0520257	[Message]	Const "*symbol*" requires an initializer.
W0520260	[Message]	Explicit type is missing ("int" assumed).
W0520301	[Message]	typedef name has already been declared (with same type).
W0520375	[Message]	Declaration requires a typedef name.
W0520494	[Message]	Declaring a void parameter list with a typedef is nonstandard.
W0520513	[Message]	A value of type "*type1*" cannot be assigned to an entity of type "*type2*".
W0520520	[Message]	Initialization with "{...}" expected for aggregate object.
W0520546	[Message]	Transfer of control bypasses initialization of: *type* "*symbol*" (declared at line *number*).
W0520549	[Message]	*Type* "*symbol*" is used before its value is set.
W0520550	[Message]	*Type* "*symbol*" was set but never used.
W0520606	[Message]	This pragma must immediately precede a declaration.
W0520609	[Message]	This kind of pragma may not be used here.
W0520618	[Message]	struct or union declares no named members.
W0520676	[Message]	Using out-of-scope declaration of *type* "*symbol*" (declared at line *number*).
W0520767	[Message]	Conversion from pointer to smaller integer.
W0520815	[Message]	Type qualifier on return type is meaningless.
W0520819	[Message]	"..." is not allowed.
W0520867	[Message]	Declaration of "size_t" does not match the expected type "*type*".

W0520870	[Message]	Invalid multibyte character sequence.
W0520902	[Message]	Type qualifier ignored.
W0520940	[Message]	Missing return statement at end of non-void "*symbol*".
W0520951	[Message]	Return type of function "main" must be "int".
W0520966	[Message]	Universal character name specifies an invalid character.
W0520967	[Message]	A universal character name cannot designate a character in the basic character set.
W0520968	[Message]	This universal character is not allowed in an identifier.
W0520993	[Message]	Subtraction of pointer types "*type name1*" and "*type name2*" is nonstandard.
W0521000	[Message]	A storage class may not be specified here.
W0521037	[Message]	This universal character cannot begin an identifier.
W0521039	[Message]	Unrecognized STDC pragma.
W0521040	[Message]	Expected "ON", "OFF", or "DEFAULT".
W0521046	[Message]	Floating-point value cannot be represented exactly.
W0521051	[Message]	Standard requires that *NAME* be given a type by a subsequent declaration ("int" assumed).
W0521053	[Message]	Conversion from integer to smaller pointer.
W0521056	[Message]	Returning pointer to local variable.
W0521057	[Message]	Returning pointer to local temporary.
W0521072	[Message]	A declaration cannot have a label.
W0521222	[Message]	Invalid error number.
W0521223	[Message]	Invalid error tag.
W0521224	[Message]	Expected an error number or error tag.
W0521297	[Message]	Constant is too large for long long; given unsigned long long type (nonstandard).
W0521422	[Message]	Multicharacter character literal (potential portability problem).
W0521644	[Message]	Definition at end of file not followed by a semicolon or a declarator.
	[Explanation]	The declaration at the end of the file lacked a semicolon to indicate its termination.
W0521649	[Message]	White space is required between the macro name "*macro name*" and its replacement text
	[Action by User]	Insert a space between the macro name and the text to be replaced.
W0523018	[Message]	A member qualified with near or far is declared.
W0523037	[Message]	#pragma section ignored
	[Explanation]	An unavailable section type is used.
W0523061	[Message]	argument is incompatible with formal parameter of intrinsic function
W0523062	[Message]	return value type does not match the intrinsic function

W0523076	[Message]	Function declarations should have prototype.
	[Explanation]	The function is declared in a different format from the prototype. A function declaration in a format different from the prototype may degrade the efficiency when passing a near pointer as an argument.
W0523077	[Message]	Called function should have prototype.
	[Explanation]	This function call uses a function type without a prototype. When the defining function has a prototype, a mismatch may occur in passing of arguments. The call being made via the function pointer means that there is no prototype for the function pointer type.
W0523079	[Message]	The function cannot be used in CC-RL. Ignored.
	[Explanation]	Not supported by the transition support function.
W0523080	[Message]	Required to follow the CC-RL format.
W0523081	[Message]	Converted to a function of the CC-RL
W0523082	[Message]	Pointer to the object of even alignment holds the odd address
W0523083	[Message]	Combination of odd address and the type is incorrect
W0523084	[Message]	"iodefine.h" should be included
W0523085	[Message]	Address of packed member.
W0530809	[Message]	const qualifier for variable "*variable-name*" differs between files.
W0530811	[Message]	Type of symbol "*symbol-name*" differs between files.
W0533003	[Message]	Shift count(*value*) is out of range.
W0533004	[Message]	Result of comparison is always *character string*.
W0533005	[Message]	Division by zero.
W0550001	[Message]	Too many arguments.
	[Action by User]	Check the actual arguments.
W0550005	[Message]	Illegal "*option*" option's symbol "*symbol*", ignored.
	[Action by User]	Check the option specification symbols.
W0550010	[Message]	Illegal displacement.
	[Explanation]	Illegal displacement in inst instruction. Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the displacement value.
W0550011	[Message]	Illegal operand (range error in immediate).
	[Explanation]	Illegal operand (range error in immediate). Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the immediate value.
W0550012	[Message]	Operand overflow.
	[Explanation]	Operand overflow. Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the operand value.

W0550013	[Message]	*register* used as register.
	[Action by User]	Check the register specification.
W0550019	[Message]	Illegal operand (immediate must be multiple of *string*).
	[Explanation]	Illegal operand (immediate must be multiple of *string*). The number is rounded down, and assembly continues.
	[Action by User]	Check the operand value.
W0561000	[Message]	Option "*option*" ignored
	[Explanation]	The option named *option* is invalid, and is ignored.
W0561001	[Message]	Option "*option1*" is ineffective without option "*option2*"
	[Explanation]	*option1* needs specifying *option2*. *option1* is ignored.
W0561002	[Message]	Option "*option1*" cannot be combined with option "*option2*"
	[Explanation]	*option1* and *option2* cannot be specified simultaneously. *option1* is ignored.
W0561003	[Message]	Divided output file cannot be combined with option "*option*"
	[Explanation]	*option* and the option to divide the output file cannot be specified simultaneously. option is ignored. The first input file name is used as the output file name.
W0561004	[Message]	Fatal level message cannot be changed to other level : "*option*"
	[Explanation]	The level of an fatal error message cannot be changed. The specification of *option* is ignored. Only errors at the information/warning/error level can be changed with the change_message option.
W0561005	[Message]	Subcommand file terminated with end option instead of exit option
	[Explanation]	There is no processing specification following the end option. Processing is done with the exit option assumed.
W0561006	[Message]	Options following exit option ignored
	[Explanation]	All options following the exit option is ignored.
W0561007	[Message]	Duplicate option : "*option*"
	[Explanation]	Duplicate specifications of *option* were found. Only the last specification is effective.
W0561008	[Message]	Option "*option*" is effective only in cpu type "*CPU type*"
	[Explanation]	*option* is effective only in *CPU type*. option is ignored.
W0561010	[Message]	Duplicate file specified in option "*option*" : "*file name*"
	[Explanation]	*option* was used to specify the same file twice. The second specification is ignored.
W0561011	[Message]	Duplicate module specified in option "*option*" : "*module*"
	[Explanation]	*option* was used to specify the same module twice. The second specification is ignored.
W0561012	[Message]	Duplicate symbol/section specified in option "*option*" : "*name*"
	[Explanation]	*option* was used to specify the same symbol name or section name twice. The second specification is ignored.
W0561013	[Message]	Duplicate number specified in option "*option*" : "*number*"
	[Explanation]	*option* was used to specify the same error number. Only the last specification is effective.

W0561014	[Message]	License manager is not installed
	[Explanation]	The license manager is not installed. Install the correct license manager.
W0561015	[Message]	Invalid parameter specified in option "*option*" : "*parameter*"
	[Explanation]	An invalid parameter was specified by "*option*". The "*parameter*" specification will be ignored.
W0561016	[Message]	The evaluation version of *version* is valid for the remaining *number* days. After that, link size limit (64 Kbyte) will be applied. Please consider purchasing the product.
W0561017	[Message]	Paid license of "*version*" is not found, and the evaluation period has expired. Please consider purchasing the product.
W0561018	[Message]	The evaluation period of "*version*" is valid for the remaining "*N*" days. After that, functional limit will be applied. Please consider purchasing the product.
W0561100	[Message]	Cannot find "*name*" specified in option "*option*"
	[Explanation]	The symbol name or section name specified in *option* cannot be found. *name* specification is ignored.
W0561101	[Message]	"*name*" in option "*option*" conflicts between symbol and section
	[Explanation]	*name* specified by *option* exists as both a section name and as a symbol name. Rename is performed for the symbol name only in this case.
W0561102	[Message]	Symbol "*symbol*" redefined in option "*option*"
	[Explanation]	The symbol specified by *option* has already been defined. Processing is continued without any change.
W0561103	[Message]	Invalid address value specified in option "*option*" : "*address*"
	[Explanation]	*address* specified by *option* is invalid. *address* specification is ignored.
W0561104	[Message]	Invalid section specified in option "*option*" : "*section*"
	[Explanation]	An invalid section is specified in *option*.
	[Action by User]	Confirm the following: (1) The "-output" option does not accept specification of a section that has no initial value. (2) The "-jump_entries_for_pic" option accepts specification of only a code section and no other sections.
W0561120	[Message]	Section address is not assigned to "*section*"
	[Explanation]	*section* has no addresses specified for it. *section* will be located at the rearmost address.
	[Action by User]	Specify the address of the section using the rlink option "-start".
W0561121	[Message]	Address cannot be assigned to absolute section "*section*" in start option
	[Explanation]	*section* is an absolute address section. An address assigned to an absolute address section is ignored.
W0561122	[Message]	Section address in start option is incompatible with alignment : "*section*"
	[Explanation]	The address of *section* specified by the start option conflicts with memory boundary alignment requirements. The section address is modified to conform to boundary alignment.
W0561123	[Message]	Section "*section*" is placed on the "*area*"
	[Explanation]	When the -selfw option is specified, the section is allocated to *area*.

W0561124	[Message]	Section "*section*" is placed on the "*area*"
	[Explanation]	When the -ocdtrw option is specified, the section is allocated to *area*.
W0561125	[Message]	Section "*section*" is placed on the "*area*"
	[Explanation]	When the -ocdhpiw option is specified, the section is allocated to *area*.
W0561126	[Message]	CRC result is placed on the "*area*"
	[Explanation]	When the -selfw option is specified, the CRC result is placed in *area*.
W0561127	[Message]	CRC result is placed on the "*area*"
	[Explanation]	When the -ocdtrw option is specified, the CRC result is placed in *area*.
W0561128	[Message]	CRC result is placed on the "*area*"
	[Explanation]	When the -ocdhpiw option is specified, the CRC result is placed in *area*.
W0561130	[Message]	Section attribute mismatch in rom option : "*section1*","*section2*"
	[Explanation]	The attributes and boundary alignment of *section1* and *section2* specified by the rom option are different. The larger value is effective as the boundary alignment of *section2*.
W0561140	[Message]	Load address overflowed out of record-type in option "*option*"
	[Explanation]	A record type smaller than the address value was specified. The range exceeding the specified record type has been output as different record type.
W0561141	[Message]	Cannot fill unused area from "*address*" with the specified value
	[Explanation]	Specified data cannot be output to addresses higher than *address* because the unused area size is not a multiple of the value specified by the space option.
W0561142	[Message]	Cannot find symbol which is a pair of "*symbol*"
	[Explanation]	The symbol which is a pair of *symbol* indicating the range of the empty area could not be found.
W0561143	[Message]	Address *start address-end address* cannot be placed on flash memory area.
	[Explanation]	The range of "*start address-end address*" is not in the flash memory area, and there is data that cannot be written by a flash programmer.
W0561150	[Message]	Sections in "*option*" option have no symbol
	[Explanation]	The section specified in *option* does not have an externally defined symbol.
W0561160	[Message]	Undefined external symbol "*symbol*"
	[Explanation]	An undefined external symbol *symbol* was referenced.
W0561181	[Message]	Fail to write "*type of output code*"
	[Explanation]	Failed to write *type of output code* to the output file. The output file may not contain the address to which *type of output code* should be output. Type of output code: When failed to write CRC code : "CRC Code"
W0561182	[Message]	Cannot generate vector table section "*section*"
	[Explanation]	The input file contains vector table *section*. The optimizing linker does not create *section* automatically.

W0561183	[Message]	[V1.06 or earlier] Interrupt number "*vector table address*" of "*section*" is defined in input file [V1.07 or later] Interrupt table address "*vector table address*" of "*section*" is defined in input file
	[Explanation]	The vector number specified by the VECTN option is defined in the input file. Processing is continued with priority given on the definition in the input file.
W0561184	[Message]	[V1.06 or earlier] Interrupt number "*vector table address*" of "*section*" is defined [V1.07 or later] Interrupt table address "*vector table address*" of "*section*" is defined
	[Explanation]	The vector number in the -debug_monitor option has already been defined in the input file or -vectn option. Continue processing with priority given to the specification of the -debug_monitor option.
W0561191	[Message]	Area of "FIX" is within the range of the area specified by "*cpu=<attribute>*" :"*<start>-<end>*"
	[Explanation]	In the cpu option, the address range of <start>-<end> specified for FIX overlapped with that specified for another memory type. The setting for FIX is valid.
W0561193	[Message]	Section "*section name*" specified in option "*option*" is ignored
	[Explanation]	*option* specified for the section newly created due to -cpu=stride is invalid.
	[Action by User]	Do not specify *option* for the newly created section.
W0561195	[Message]	Read only data memory domains differ in "*file*"
	[Explanation]	"*file*" with a different memory model is input.
	[Action by User]	Check if the compiler options are correct.
W0561200	[Message]	Backed up file "*file1*" into "*file2*"
	[Explanation]	Input file *file1* was overwritten. A backup copy of the data in the previous version of *file1* was saved in *file2*.
W0561300	[Message]	Option "*option*" is ineffective without debug information
	[Explanation]	There is no debugging information in the input files. The "*option*" has been ignored.
	[Action by User]	Check whether the relevant option was specified at compilation or assembly.
W0561301	[Message]	No inter-module optimization information in input files
	[Explanation]	No inter-module optimization information is present in the input files. The optimize option has been ignored.
	[Action by User]	Check whether the goptimize option was specified at compilation or assembly.
W0561302	[Message]	No stack information in input files
	[Explanation]	No stack information is present in the input files. The stack option is ignored. If all input files are assembler output files, the stack option is ignored.
W0561305	[Message]	Entry address in "*file"* conflicts : "*address*"
	[Explanation]	Multiple files with different entry point addresses are input.
W0561310	[Message]	"*section*" in "*file*" is not supported in this tool
	[Explanation]	An unsupported section was present in *file*. *section* has been ignored.

W0561311	[Message]	Invalid debug information format in "*file*"
	[Explanation]	Debugging information in *file* is not dwarf2. The debugging information has been deleted.
W0561320	[Message]	Duplicate symbol "*symbol*" in "*file*"
	[Explanation]	The symbol named *symbol* is duplicated. The symbol in the first file input is given priority.
W0561322	[Message]	Section alignment mismatch : "*section*"
	[Explanation]	Sections with the same name but different boundary alignments were input. Only the largest boundary alignment specification is effective.
W0561323	[Message]	Section attribute mismatch : "*section*"
	[Explanation]	Sections with the same name but different attributes were input. If they are an absolute section and relative section, the section is treated as an absolute section. If the read/write attributes mismatch, both are allowed.
W0561324	[Message]	Symbol size mismatch : "*symbol*" in "*file*"
	[Explanation]	Common symbols or defined symbols with different sizes were input. A defined symbol is given priority. In the case of two common symbols, the symbol in the first file input is given priority.
W0561325	[Message]	Symbol attribute mismatch : "*symbol*" : "*file*"
	[Explanation]	The attribute of *symbol* in *file* does not match the attribute of the same-name symbol in other files.
	[Action by User]	Check the symbol.
W0561326	[Message]	Reserved symbol "*symbol*"is defined in "*file*"
	[Explanation]	Reserved symbol name *symbol* is defined in *file*.
W0561328	[Message]	Section "*section*" border mismatch in "*file*"
	[Explanation]	The attribute of the allocation border of *section* of *file* differs from the others.
W0561329	[Message]	Devided option "*start*" at beginning of Section "*section*"
	[Explanation]	Since it is possible that the 64-K or (64K-1) byte boundary may be exceeded by optimization, the -start option specification is split at the start of *section*.
W0561331	[Message]	Section alignment is not adjusted : "*section*"
	[Explanation]	Sections with the same name but different boundary alignment values were input. Only the largest boundary alignment specification is effective. The alignment condition at input may not be satisfied.
W0561402	[Message]	Parentheses specified in option "start" with optimization
	[Explanation]	Optimization is not available when parentheses "( )" are specified in the start option. Optimization has been disabled.
W0561410	[Message]	Cannot optimize "*file*"-"*section*" due to multi label relocation operation
	[Explanation]	A section having multiple label relocation operations cannot be optimized. Section *section* in *file* has not been optimized.
W0561520	[Message]	"*user option byte*/*control value for the on-chip debug*" in "*section*" created by device file
W0561521	[Message]	Cannot generate section "*section*"
	[Action by User]	Check the option during compiling and the section specification of a source file.

W0561531	[Message]	Option "*option*" ignored because Device file with "*tag*" is required
	[Action by User]	Check the device file.
W0580304	[Message]	"%s" option is specified more than once. The latter is valid.
	[Explanation]	The same option was specified more than once.
	[Action by User]	The last specification will be valid.
W0580306	[Message]	the folder "%s" specified by the "-I" option is not found.
	[Explanation]	The folder specified for the -I option was not found.
	[Action by User]	Check the specification of the -I option.
W0591300	[Message]	Command parameter specified twice "*option-name*"
	[Explanation]	The option that can be specified only once is specified more than once. Check the option you want to enable.
W0591301	[Message]	"*option-name*" option ignored
	[Explanation]	The specification of "*option-name*" is ignored.

11. CAUTIONS

This chapter explains the points to be noted when using the CC-RL.

11.1 Cautions Regarding Compiler

This section explains the cautions regarding the compiler.

11.1.1 Indirect reference of pointer

When an odd value is set to the pointer which points to a type with a 2-byte alignment condition, indirect reference to that pointer results in invalid operation.

This is because, at indirect reference of a pointer, the compiler should use an instruction suitable for the alignment condition of the type that pointer points to.

Example

```
int __near * volatile x;
int func(void){
        x = (int __near *)0xfaa1;
        return *x;                  //Since the compiler outputs an instruction
                                    //suitable for the 2-byte alignment condition to
                                    //*x, *x references the data pointed to by 0xfaa0
                                    //when the code is executed
}
```

11.1.2 Register access via pointer

When a register used in the generated code of the compiler is accessed via a pointer, invalid operation may occur.

This is because the compiler generates codes on the assumption that a register in use will not be accessed via a pointer.

When the following registers are accessed via a pointer, program operation cannot be guaranteed.

- General registers belonging to a register bank that may be used
- SP
- PSW
- CS
- ES
- PMC

Example

```
*(int __near *)0xfef8 = 7;      //Since 0xffef8 is the address of register AX in bank
                                //0, operation of a program including this
                                //description and also using bank 0 is not guaranteed
```

11.1.3 Function calling

When a function in a file is called, if the definition of that function is not in the file or the function definition is included but the function is called before that function definition, it is recommended to declare the prototype of the function before the function is called.

This is because if the type of the parameters is unknown at the function call, there is a possibility that the compiler will call the function as a different type from the parameter type in the function definition[Note], and the execution result of the program will be invalid.

Note An arithmetic type smaller than the int type or unsigned int type is treated as the int type, or the unsigned int type when it cannot be represented with the int type, and the float type is treated as the double type. For others, see "(7) Default argument promotions".

Example 1. Program with an incorrect execution result

```
AAA.c:
        void func(char a, char b);  //The two paramters receive their values from
                                    //registers A and X
BBB.c:
        extern void func();         //Not a prototype declaration because the
                                    //parameter types are not described
        void main (void)
        {
                char    x=1, y=1;
                func(x, y);         //After x and y are each extended to the int type
                                    //and allocated to registers AX and BC, the
        }                           //function call is made
```

Example 2. Program with a correct execution result

```
AAA.c:
        void func(char a, char b);          //The two paramters receive their values
                                            //from registers A and X
BBB.c:
        extern void func(char a, char b);   //Prototype declaration
        void main (void)
        {
                char    x=1, y=1;
                func(x, y);                 //After the two parameters are allocated
        }                                   //to registers A and X, the function call
                                            //is made
```

If compiler option "-refs_without_declaration" is specified, whether there is a function declaration can be checked.
If a function without a prototype is called, an error will occur.

11.1.4 Data flash area

The CC-RL does not output code to the data flash area.
Use assembler code for access to the data flash area.
In case of writing codes in C language, use 8-bit data to access the area.

11.1.5 Function definitions in K&R format (formal parameters of _Bool type)

When the function definition that includes a _Bool-type parameter is written in the K&R format, an assembly code that assigns the argument value to the _Bool-type parameter without change is generated. Therefore, if the _Bool-type argument has a value other than 0 or 1, a value which is neither 0 nor 1 will be set to the parameter.

Example

```
void sub();
signed char c;
void func(void) {
        sub(2);
}

void sub(b)                     // 2 is set to parameter b.
_Bool b;
{
        if (b == 0 || b == 1) {
                c = b;
        } else {
                c = -1;         // -1 is set to c.
        }
}
```

(Workaround)
Write the function definition including a _Bool-type parameter and the function declaration in the function prototype format.

```
void sub(_Bool);                // Function prototype
signed char c;
void func(void) {
        sub(2);
}

void sub(_Bool b)               // Function prototype
{
        if (b == 0 || b == 1) {
                c = b;
        } else {
                c = -1;
        }
}
```

11.1.6 MISRA2004 check (rule number 10.1)

Unnecessary message may be output for statements with enumerated-type variables, return statements with enumerated-type return values, or statements with enumerators in a file satisfying the following conditions (1), (2), and (3).

(1) Option -signed_char is not specified and enumerated-type definitions whose range of enumerator value is within a range of 0 to 255 are included,
or
option -signed_char is specified and enumerated-type definitions whose range of enumerator value is within a range of -128 to 255 are included.

(2) Statements with enumerated-type variables, return statements with enumerated-type return values, or statements with enumerators are included.

(3) MISRA check option against rule 10.1 is specified.

Example

```
typedef enum E { E1 = 0, E2, E3 } etype;
etype func( void );
etype evar;
etype func(void)
{
        evar = E1;                      // Message against rule 10.1 is output.
        return E1;                      // Message against rule 10.1 is output.
}
```

(Workaround)
Cast enumerators to enumerated type.

```
typedef enum E { E1 = 0, E2, E3 } etype;
etype func( void );
etype evar;
etype func(void)
{
        evar = (etype)E1;               // Cast enumerator E1 to etype
        return (etype)E1;               // Cast enumerator E1 to etype
}
```

11.1.7 Extended language specifications which needs the device file

A device file must be specified when the following functions are used.

- #pragma callt or __callt
- #pragma saddr or __saddr
- const variable with the __near attribute

If a device file is not specified, there is a possibility that an incorrect code will be generated.

11.1.8 Controlling the Output of Bit Manipulation Instructions [V1.04 or later]

To output bit manipulation instructions without using intrinsic functions, satisfy all conditions shown below.

(a) A constant value is assigned.

(b) The value is assigned to a single-bit bit field of the char/unsigned char/signed char/_Bool type in the near area.

(c) The bit field where the value is assigned is qualified with volatile.

For a variable qualified with volatile, the compiler does not output a bit manipulation instruction when a value is assigned to the variable or when the variable is read unless the other conditions are satisfied. For a variable that is not qualified with volatile, the compiler outputs a bit manipulation instruction according to the specified optimization settings.

Example

```
volatile struct {
  unsigned char bit0:1;
  unsigned int  bit1:1;
} data;

void func(void) {
  data.bit0 = 1;  /* A bit manipulation instruction is output. */
  data.bit1 = 1;  /* No bit manipulation instruction is output. */
}
```

11.2 Cautions Regarding Library and Startup

This section explains the cautions regarding the library and startup.

11.2.1 Setting of Processor Mode Control Register (PMC)

PMC should be set only once at initial setting. Rewriting PMC other than at initial setting is prohibited.

11.2.2 Label whose value is determined by the linker

When the following labels used at startup are defined, operation becomes erroneous. Since the optimizing linker will determine these labels, definitions are unnecessary.
__STACK_ADDR_START, __STACK_ADDR_END, __RAM_ADDR_START, __RAM_ADDR_END

11.2.3 Options necessary at assembling the startup file

When startup is directly assembled without intervening the compiler driver, specify the following options.
-define=__RENESAS_VERSION__=0x01000000 ; When CC-RL V1.00 is used
-define=__RENESAS_VERSION__=0x01010000 ; When CC-RL V1.01 is used

11.2.4 Usage Restriction of Standard Library Function Name

Among the functions included in the standard library, the following function names must not be described as user variable names or user function names in the program.
This is because these are normally included in the runtime library which needs to be linked.

- memcpy, memset

11.2.5 Error in standard library functions

- If the number of digits of floating-point numbers is high in printf, sprintf, vprintf, vsprintf, scanf, sscanf, atof, atoff, strtod, or strtof, the error is enlarged.
- If the absolute value of an exponential is high in pow, the error is enlarged.

11.2.6 Definition of comparison functions bsearch and qsort in K&R format

When the definition of comparison function bsearch or qsort is written in the K&R format, the caller side of comparison function bsearch or qsort passes the argument as a near pointer but the defining side of the comparison function generates an assembler code in which the formal parameter is received as a far pointer. Therefore, the program will not operate correctly.

Example

```
#include <stdlib.h>
int table[5] = {0, 1, 2, 3, 4}, key = 3, *ret;
int compare();
void func(void) {
        ret = bsearch(&key, table, 5, sizeof(int), compare);
}
int compare(i, j)
int *i, *j;
{
        if (*i > *j) {
                return 1;
        }
        else if (*i < *j) {
                return -1;
        }
        return 0;
}
```

(Workaround)
Write the definition and declaration of comparison function bsearch or qsort in the function prototype format.

```
#include <stdlib.h>
int table[5] = {0, 1, 2, 3, 4}, key = 3, *ret;
int compare(int *, int *);      // Function prototype

void func(void) {
        ret = bsearch(&key, table, 5, sizeof(int), compare);
}

int compare(int *i, int *j)     // Function prototype
{
        if (*i > *j) {
                return 1;
        }
        else if (*i < *j) {
                return -1;
        }
        return 0;
}
```

11.2.7 Initialization of Stack Area at Startup [V1.07 or earlier]

The processing for initializing the stack area is commented out.To perform parity error detection by reading uninitialized RAM, enable the processing for initializing the stack area.

11.2.8 Specifying standard library functions when C99 standard is specified by an individual option

When calling standard library functions complying with the C99 standard from a source program based on the C99 standard with the C99 standard specified individually by the -lang option, use a link option to specify usage of the standard library functions for the C99 standard.

11.3 Cautions Regarding Assembler

This section explains the cautions regarding the assembler.

11.3.1 Assembler driver

When multiple input files are specified for the assembler driver (asrl), execution may be terminated abnormally.

11.3.2 .DB8 directive

Forward reference in a .DB8 directive is prohibited, no error occurs even when a .DB8 directive is forward referenced. Define the symbol before it is a .DB8 directive in order to prevent forward reference.

Example

```
        .DB8    SYM1            ; Forward reference is prohibited
SYM1    .EQU    0x12345         ; Define the symbol after it is a .DB8 directive

SYM2    .EQU    0x12345         ; Define the symbol before it is a .DB8 directive
        .DB8    SYM2            ; Backward reference can be described.
```

11.3.3 Bit symbols

- When a symbol that was defined with a bit position specifier in an .EQU directive is forward referenced, no error occurs but the result is not correct. Define the bit symbol before it is referenced in order to prevent forward reference.

Example

```
OFFSET   CODE                  NO  SOURCE STATEMENT
00000000                        1
00000000 22220007               2        .DB4    BSYM
00000004                        3  BSYM  .EQU    0x2222.7
```

- Operation between bit symbols does not cause an error but the result is not correct. Do not write code for operation between bit symbols.

Example

```
OFFSET   CODE                  NO  SOURCE STATEMENT
00000000                        1
00000000                        2
00000000                        3  BSYM1 .EQU    0x2222.5
00000000                        4  BSYM2 .EQU    0x3333.7
00000000                        5
00000000 11110002               6        .DB4    BSYM2 - BSYM1
```

11.3.4 .ALIGN directive

Even if an .ALIGN directive is written in an absolute section (section whose start address is specified), ALIGN becomes invalid.

Example

```
        .CSEG TEXT
        .ORG    0x1
        .ALIGN 2        ; Invalid
_LABEL:
```

11.3.5 Separation operators

[V1.01 only]
When a separation operator is applied to a relocatable term, though only the parameters enclosed in parentheses ( ) should be subject to operation under normal conditions, the value (including a symbol) following the parentheses ( ) will also be included in the operation target.
The relevant separation operators are as follows:

- HIGH, LOW, HIGHW, LOWW, MIRHW, MIRLW, SMRLW

Example

```
_L:
        mov A, !HIGH(_L) + 0x44         ; Interpreted as "!HIGH(_L+0x44)"
```

[V1.02 or later]
When a nested separation operator is applied to a relocatable term and addition and subtraction between the separation operator and constant are written, though only the parameters enclosed in parentheses ( ) should be subject to operation under normal conditions, operation will be performed with the constant in parentheses ( ) written after the parentheses ( ). The relevant separation operators are as follows:

- HIGH,LOW,HIGHW,LOWW,MIRHW,MIRLW,SMRLW

Example

```
_L:
        mov A, !HIGH(LOWW(_L+0x44)+0x55)  ; Interpreted as "!HIGH(LOWW(_L+0x44)+0x55"
```

11.3.6 Predefined macro enabled in an assembly source file

When invoking the assembler via the compile driver, predefined macro __RENESAS_VERSION__ is enabled in an assembly source file.
The macro is enabled in an assembly source file even when "-U __RENESAS_VERSION__" is specified.

To disable the macro in an assembly source file, perform either of the followings.

- Specify "-asmopt=-undefine=__RENESAS_VERSION__".
- Invoke the assembler directly.

11.3.7 An option depending on the order of specification of options

Option "-no_warning" has no effect if the order of specification of options is inappropriate.

Example "-no_warning" has no effect in the following case.
-define=xxxx -undefine=xxxx -no_warning=50649

To suppress a warning message, perform either of the followings.

- When the assember is invoked via the compile driver,
  - specify compile option "-no_warning_num=50649", or
  - specify "-asmopt=-no_warning=50649" before other options.
- When the assembler is invoked directly,
  - specify "-no_warning" before the other options.

    Example "-no_warning" has an effect in the following case.
    -no_warning=50649 -define=xxxx -undefine=xxxx

11.4 Cautions Regarding Linker

This section explains the cautions regarding the linker.

11.4.1 -strip option

Multiple input files cannot be specified with the strip option. Specify one file at a time.

11.4.2 -memory option

When all of the following conditions are satisfied, execution may be terminated abnormally. Do not specify the -memory=low option.

- The -memory=low option has been specified at linkage.
- The -nooptimize option has not been specified at linkage.

(Workaround)
Do not specify the -memory=low option.

11.4.3 Overwrite of variable/function information file

After the variable/function information file that was output by the -vfinfo option specification is edited, specifying the same file name in the -vfinfo option at the second linkage will overwrite the variable/function information file.
Save the edited variable/function information file under another name, do not specify the -vfinfo option after the file has been edited, or change the output file name of the -vfinfo option.

11.4.4 Allocation of sections

When the section automatic allocation option (-auto_section_layout) is not specified, a section is not automatically allocated to the area (address range) corresponding to the relocation attribute specified by the .SECTION, .CSEG, or .DSEG directive. Allocation that avoids exceeding the (64K-1) byte boundary is also not carried out.
Specify a suitable address for allocating each section using the -start option of the linker.

11.4.5 Variable/function information file that may cause a compile error

When -vfinfo option is specified at link time for a program containing global functions with __inline keyword, and then the program is compiled with the generated variable/function information file, a compile error may occur.
That happens when variable/function information file contains "#prgma __callt" directive for the above function.

To avoid an error, perform either of the followings.

- Edit the variable/function information file and comment out or remove the #pragma directive for the function.
- Remove __inline keyword.
- Do not specify -vfinfo option.

11.4.6 Error output regarding an address not in the saddr access range

When the -dev option is specified, an error will be output for an address outside of the saddr access range. However, an error will not be output when the -dev option is not specified.

Example >ccrl -cpu=S3 file.asm -lnkopt=-start=.text/100,.bss/f9f34

```
        mov     a,!sym
        mov     a,sym   ; When the -dev option is specified, an error will be output.

        .section .bss, bss
sym:    .DS     (1)
```

11.4.7 Version of Compiler Package

When using an optimizing linker, use one provided with the same compiler package used to generate all object files, relocatable files, and library files that are to be input. An optimizing linker provided with a newer compiler package can also be used.

When using standard library functions and runtime library functions, use those provided with the same compiler package as the optimizing linker in use.

A. QUICK GUIDE

This chapter explains the programming method and how to use the extended language specifications for more efficient use of the CC-RL.

A.1 Variables (C Language)

This section explains variables (C language).

A.1.1 Allocating to sections accessible with short instructions

The CC-RL provides an attribute among far, near, or saddr to variables or constants, and allocates them to areas with the same names as their attributes; far area, near area, or saddr area.

Since the size of instructions to access variables or constants in these area becomes smaller in this order, the code size can be reduced by specifying the near attribute (or saddr attribute) for variables or constants and then allocating them to the near area (or saddr area).

As the size of the area also becomes smaller in this order, it is necessary to prioritize which area to allocate each variable for a large total size of variables or constants.

The specification method of each attribute, each area, and its allocation method are described in detail in the following.

(1) Methods for specifying the attribute for variables or constants
Options or keywords are used to specify the attribute for variables or constants.
The following four specification methods are available and the priority gets higher in this order.
For details, see each option or keyword and "2.6.6 Relationship with near and far".

(a) -cpu option
The default attribute for variables or constants is always the near attribute.

(b) -memory_model option
The default attribute for variables or constants is always the near attribute.

(c) -far_rom option
The ROM data has far attribute by default.

(d) __near/__far/_saddr keyword
A variable has the attribute specified by each keyword.

Example 1. When the -far_rom option is not specified

```
long                x;      //near attribute
long __near         y;      //near attribute
long __far          z;      //far attribute
const int           c;      //near attribute
const int __near    d;      //near attribute
const int __far     e;      //far attribute
char __saddr        s;      //saddr attribute
```

Example 2. When the -far_rom option is specified

```
long                x;      //near attribute
long __near         y;      //near attribute
long __far          z;      //far attribute
const int           c;      //far attribute
const int __near    d;      //near attribute
const int __far     e;      //far attribute
char __saddr        s;      //saddr attribute
```

(2) Priority for allocating variables or constants to each area
In general, if variables with higher static reference counts are allocated to an area accessible with a small size of instructions, the code size of the entire program gets smaller.
However, an area accessible with a small size of instructions needs to be used efficiently because the size of the area is also small.
Therefore, it is necessary to maximize the reduction efficiency for one byte of the area, namely, it is preferable to allocate variables whose value obtained by dividing the static reference count of the variable with the variable size is larger to an area accessible with a smaller size of instructions.
Accordingly, the priority is conceived as follows

Priority = static reference count of the variable / size of the variable

Here, the static reference count of the variable indicates the reference count in the object code, and this will differ from the reference count in the C source program.
In the RL78, instructions for variable reference have only a 1-byte or 2-byte access width. Thus, for example, two instructions are needed to reference a 4-byte variable.
Therefore, one reference to a 4-byte variable in a C source program corresponds to two references in the object code.
The reference count in the object code can be shown by specifying -show=reference option to linker.

(3) Area
The areas placing variables or constants is divided into three, based on the address range.
The far area is the entire area including the saddr area or near area.

(a) saddr area
The area from addresses 0xFFE20 to 0xFFF1F is called the saddr area.
The saddr-attribute variables located in this address range are accessed by instructions with the smallest size.
The saddr area includes some of the special function registers (SFRs) and all general-purpose registers.
Variables cannot be allocated to the SFRs or general-purpose registers.

(b) near area
The area from addresses 0xF0000 to 0xFFE1F is called the near area.
This area includes internal RAM, mirror region, and data flash memory.
The near-attribute variables located in this address range are accessed by instructions with the second smallest size after the saddr area.
The near area also includes the extended special function registers (2nd SFRs).
Variables cannot be allocated to the 2nd SFRs.

(c) far area
The entire area including the saddr area or near area is called the far area.
This area includes RAM other than internal RAM and the constant area in code flash memory other than the mirror region.
The far-attribute variables located in this address range are accessed by instructions with the largest size.

(4) Method for allocating variables or constants to each area
Variables or constants are allocated to each area by using link options to specify the allocation address of the section including each variable or constant.

Example 1. The default sections for the allocation of saddr variables are .sbss and .sdata.
.sbss is the section for the allocation of saddr-attribute variables without initial values and .sdata is the section for the allocation of saddr-attribute variables with initial values.
When you create a ROM image, allocate the .sdata section to a desired address and give an appropriate name, .sdataR in the example below, to the destination section for transfer in RAM in the case of the transfer of data from the .sdata section to RAM.

```
-rom=.sdata=.sdataR
-start=.sdata/5000
-start=.sbss,.sdataR/ffe20
```

Example 2. The default sections for allocation to the near area are .bss, .data and .const.
.bss is the section for the allocation of near-attribute variables without initial values and .data is the section for the allocation of near-attribute variables with initial values.
When you create a ROM image, allocate the .data section to a desired address and give an appropriate name, .dataR in the example below, for the destination section for transfer in RAM in the case of the transfer of data from the .data section to RAM.

```
-rom=.data=.dataR
-start=.const/2000
-start=.data/4000
-start=.bss,.dataR/fe000
```

Example 3. The default sections allocated to the far area are .bssf, .dataf and .constf.
.bssf or .dataf is normally not used when only internal RAM is to be used as RAM.
.constf is a section for allocating constants with far attribute.
.constf is allocated to code flash memory.

```
-start=.constf/6000
```

A.1.2 Defining variables for use during both ordinary and interrupt processing

Specify volatile qualifier to the variables that are to be used during both ordinary and interrupt processing.
When a variable is defined with the volatile qualifier, the variable is not optimized. The reference to the volatile variables is compiled to the code which always read the value from memory, and the assignment to the volatile variables is compiled to the code which always write the value to memory. The access order, access width, and access count of volatile variables are not changed. A variable for which volatile is not specified may be assigned to a register as a result of optimization and the code that loads the variable from the memory may be deleted. When the same value is assigned twice to a variable for which volatile is not specified, the latter assignment instruction may be deleted as a result of optimization because it is interpreted as a redundant instruction.

Example 1. Example of source and output code image when volatile is not specified
If variables a and b are not specified with the volatile quantifier, they may be assigned to registers, and may be optimized.

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movw    ax, !LOWW(_a)
        xor     a, #0x80
        cmpw    ax, #0x8001
        movw    ax, !LOWW(_b)
        bnc     $.BB@LABEL@1_3
.BB@LABEL@1_1:          ; bb1
        incw    ax
.BB@LABEL@1_2:          ; bb9
        incw    ax
        movw    !LOWW(_b), ax
        ret
.BB@LABEL@1_3:          ; bb3
        incw    ax
        br      $.BB@LABEL@1_1
```

Example 2. Source and output code when volatile has been specified
If the volatile qualifier is specified for variables a and b, the output code is such that the values of these variables are always read from and written to memory when values are assigned to them.
When volatile is specified, the code size increases compared with when volatile is not specified because the memory has to be read and written.

```
volatile int a;
volatile int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movw    ax, !LOWW(_a)
        xor     a, #0x80
        cmpw    ax, #0x8001
        bnc     $.BB@LABEL@1_2
.BB@LABEL@1_1:          ; bb1
        onew    ax
        br      $.BB@LABEL@1_3
.BB@LABEL@1_2:          ; bb3
        movw    ax, #0x0002
.BB@LABEL@1_3:          ; bb3
        addw    ax, !LOWW(_b)
.BB@LABEL@1_4:          ; bb9
        movw    !LOWW(_b), ax
        incw    !LOWW(_b)
        ret
```

A.1.3 Defining const pointer

The pointer is interpreted differently depending on the "const" specified location.

- const char *p;
  This indicates that the object (*p) which pointer p points to cannot be rewritten.
  The pointer itself (p) can be rewritten.
  Therefore each statement in the following example is judged as written in the comment and the pointer itself is allocated to RAM (.data etc.).

```
*p = 0;     /*error*/
p = 0;      /*correct*/
```

- char *const p;
  This indicates that the pointer itself (p) cannot be rewritten.
  The object (*p) which pointer p points to can be rewritten.
  Therefore each statement in the following example is judged as written in the comment and the pointer itself is allocated to ROM (.const/.constf).

```
*p = 0;     /*correct*/
p = 0;      /*error*/
```

- const char *const p;
  This indicates that neither the pointer itself(p) and the object (*p) which pointer p points to can be rewritten.
  Therefore each statement in the following example is judged as written in the comment and the pointer itself is allocated to ROM (.const/.constf)).

```
*p = 0;     /*error*/
p = 0;      /*error*/
```

A.2 Functions

This section explains functions.

A.2.1 Changing area allocation

To change a section name in the program area, use the #pragma section directive as shown below.

```
#pragma section text  "section name"
```

When a text-attribute section is created by #pragma section directive, the resulting section name becomes the "section name" in the directive appended with "_n" for a near section and the one appended with "_f" for a far section.
For a case in which section type "text" is omitted in the directive or a case in which there are both directives with section type and without it, see "Changing compiler output section name (#pragma section)".

Specify the start address of the section with the -start option, as follows.

```
-start=Mytext_n/1000
```

Specify the address as a base-16 number. If the address is not specified, it will be assigned from address 0.
The -start option is a link option. For details, see "-STARt option".

A.2.2 Embedding assembler instructions

With the CC-RL assembler instructions can be described in the following formats within C source programs. This treats the function itself as assembler instruction, and performs inline expansion at the call site.

- #pragma directive

```
#pragma inline_asm func
static int func(int a, int b) {
        /*Assembler instruction*/
}
```

For details, see "Describing assembler instruction (#pragma inline_asm)".

A.2.3 Executing a program in RAM

A program allocated in ROM can be transferred to RAM and executed in RAM.
The attribute of the program to be transferred should be the far attribute.
The default text section with a far attribute is .textf.
If not the whole but only part of the .textf section is to be executed from RAM, use #pragma section to change the section name and specify that section name for the -rom option. After having been transferred from ROM to RAM, the section can be executed from RAM.

Example If an interrupt occurs, f1 and f2 are transferred to RAM and executed in RAM.

- File : ram.c

```
#include "iodefine.h"
#pragma section text    ram_text

__far void f1(char) {...}
__far int f2(int) {...;f1(x);...}

#pragma section
#pragma interrupt       inthandler (vect=INTP0)

void inthandler(void){
    /*Program is transferred from the ram_text_f section to the ram_text_fR section*/
        unsigned char __far *dst, *src;
        src = __sectop("ram_text_f");
        dst = __sectop("ram_text_fR");
        while (src < __secend("ram_text_f")) {
                *dst++ = *src++;
        }
    /*Call the program that was transferred to RAM*/
        f2(1);
}
```

- Link option

```
-rom=ram_text_f=ram_text_fR
-start=ram_text_f/3000
-start=ram_text_fR/ff000
```

A.3 Variables (Assembly Language)

This section explains variables (Assembly language).

A.3.1 Defining variables with no initial values

Use the .DS directive in a section with no initial value to allocate area for a variable with no initial value.

```
[label:]        .DS     size
```

In order that it may be referenced from other files as well, it is necessary to declare the label with the .PUBLIC directive.

```
.PUBLIC Symbol name
```

Example Defining variables with no initial values

```
        .DSEG   sbss
        .PUBLIC _val0       ;Sets _val0 as able to be referenced from other files
        .PUBLIC _val1       ;Sets _val1 as able to be referenced from other files
        .PUBLIC _val2       ;Sets _val2 as able to be referenced from other files
        .ALIGN  2           ;Aligns _val0 to 2 bytes
_val0:
        .DS     4           ;Allocates 4 bytes of area for val0
_val1:
        .DS     2           ;Allocates 2 bytes of area for val1
_val2:
        .DS     1           ;Allocates 1 byte of area for val2
```

A.3.2 Defining variable with initial values

To allocate a variable area with a default value, use the .DB directives/.DB2 directives/.DB4 directives in the section with the default value.
See "8.4 Creating ROM Images" for variable with initial values.

- 1-byte values

```
[label:]        .DB value
```

- 2-byte values

```
[label:]        .DB2 value
```

- 4-byte values

```
[label:]        .DB4 value
```

In order that it may be referenced from other files as well, it is necessary to declare the label with the .PUBLIC directive.

```
.PUBLIC Symbol name
```

Example Defining variable with initial values

```
        .DSEG   sdata
        .PUBLIC _val0       ;Sets _val0 as able to be referenced from other files
        .PUBLIC _val1       ;Sets _val1 as able to be referenced from other files
        .PUBLIC _val2       ;Sets _val2 as able to be referenced from other files
        .ALIGN  2           ;Aligns _val0 to 2 bytes
_val0:
        .DB4    100         ;Allocates a 4-byte area for _val0, and stores 100 in it
_val1:
        .DB2    10          ;Allocates a 2-byte area for _val1, and stores 10 in it
_val2:
        .DB     1           ;Allocates a 1-byte area for _val2, and stores 1 in it
```

A.3.3 Defining const data

To define a const, use the .DB directives/.DB2 directives/.DB4 directives within the .const or .constf section.

- 1-byte values

```
[label:]        .DB value
```

- 2-byte values

```
[label:]        .DB2 value
```

- 4-byte values

```
[label:]        .DB4 value
```

Example Defining const data

```
        .CSEG   const
        .PUBLIC _p          ;Sets _p as able to be referenced from other files
        .ALIGN  2           ;Aligns _val0 to 2 bytes
_p:
        .DB2    10          ;Allocates a 2-byte area for _p, and stores 10 in it
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug 01, 2014	-	First Edition issued
1.01	Feb 01, 2015	396, 397, and others	The relocation attributes of the assembler SBSS_BIT, BSS_BIT, and BIT_AT are added.
		405, 406, and others	The assembler directive .BSEG is added.
		420, and others	The assembler directive .DBIT is added.
		132	The following description is supported as the assembler transition support function. No BSEG relocation attribute, BSEG UNIT, BSEG AT, and DBIT
		133	Multiple operands can be written using the following directives of the assembler transition support function in order to enhance transition support. DB, DW, DG
		133	Writing of "$INCLUDE(" (followed immediately by a space (hereafter referred to as ∆)) is supported in order to enhance transition support. "$INCLUDE(∆ a.inc)" is the same meaning as "$INCLUDE(a.inc)".
		272	The following macro is added in the assembler. __RENESAS_VERSION__: 0xXXYYZZ00 when the version is V.XX.YY.ZZ
		471	.SLIB and .RLIB, which are sections for code of the standard libraries and runtime libraries, are added.
		544, 546, 551, 553	The following library functions and macros are added. printf_tiny, sprintf_tiny, __PRINTF_TINY__
		516, 517	Processing of NaN in library functions ldexp and ldexpf is changed as follows: When the input argument is a Not-a-Number (NaN), the NaN is returned and macro EDOM is set to global variable errno.
		272	The following macros are added. __CNV_CA78K0R__, __CNV_NC30__, __CNV_IAR__, __BASE_FILE__ The following macro is corrected so that it is valid only when the -ansi option is specified. __STDC_VERSION__
		272	The errors in the descriptions of the following macros are corrected. No value is set. -> Decimal constant 1 __RENESAS__, __RL78__, __RL78_S1__, __RL78_S2__, __RL78_S3__, __RL78_SMALL__, __RL78_MEDIUM__, __CCRL__, __CCRL, __DBL4, __DBL8, __SCHAR, __UCHAR, __SBIT, __UBIT, __FAR_ROM__
		277	A variadic macro is enabled.
		291, 292, and others	Keyword __callt is added.

Rev.	Date	Description	
		Page	Summary
1.01	Feb 01, 2015	326, 327	#pragma callt is added.
		325	#pragma saddr is added.
		328, 305, 314, 323	Binary constants can be written in a C language syntax or the following #pragma directives. #pragma interrupt (the vect parameter) #pragma rtos_interrupt (the vect parameter) #pragma address (address)
		330	Embedded functions __get_psw and __set_psw are added.
		314, and others	The vect specification of #pragma rtos_interrupt is changed to an omittable writing method, and the specifications when vect has not been specified are added.
		305, 309, 314	The specifications of #pragma interrupt and #pragma rtos_interrupt are changed. - When vect is specified, the interrupt handler is forced to have the __near attribute. - When vect is not specified, the __near or __far attribute of the interrupt handler is not changed. The specifications of #pragma interrupt_brk are changed. - The interrupt handler is forced to have the __near attribute.
		283	The error in the explanation regarding the allocation method of the bit field is corrected.
		311	The specification is changed so that "." (dot) can be used at the beginning of a changed section only when the section type is specified in #pragma section.
		309	Descriptions on generation of the DI instruction are deleted from the explanation regarding the nested interrupt enable specification of #pragma interrupt_brk.
		77, 264	gb2312 is changed to gbk in a multibyte character of the C language and the specification format of the -character_set option.
		24, 71	The compile option -pack is added.
		25, 92 -103	The compile option -convert_cc is added.
		24, 79, 81, 82	The following compile options are made usable in the Professional Edition. -misra2004, -ignore_files_misra, -check_language_extension
		748	The error in the restriction on using the standard library function name is corrected.
		176	The link option -VFINFO is added.
		194	The link option -AUTO_SECTION_LAYOUT is added.
		195	The link option -DEBUG_MONITOR is added.
		196	The link option -RRM is added.
		197	The link option -SELF is added.
		198	The link option -SELFW is added.
		199	The link option -OCDTR is added.

Rev.	Date	Description	
		Page	Summary
1.01	Feb 01, 2015	200	The link option -OCDTRW is added.
		201	The link option -OCDHPI is added.
		202	The link option -OCDHPIW is added.
		205	The link option -CHECK_DEVICE is added.
		206	The link option -CHECK_64K_ONLY is added.
		207	The link option -NO_CHECK_SECTION_LAYOUT is added.
		179, 180	A change is made so that the information on the structure and union members is output to the link map file when the link option -show=struct is specified.
		469	The following symbols which are generated when a link option is specified are added. __STACK_ADDR_START, __STACK_ADDR_END, __RAM_ADDR_START, __RAM_ADDR_END, .monitor1, .monitor2
		721	A warning will be output if possible when an odd address is indirectly referenced by a type of two bytes or more.
		621	The specifications for setting the stack area at startup are changed.
		744	20 cautions are added.
		321, 322	Examples are added to #pragma inline_asm.
		438, 439	[Detailed description] of the .EXITM and .EXITMA directives is changed.
		415, 416, and others	Examples are added to assembly language specifications.
1.02	Sep 14, 2015	23, 34, and others	The compile option -g_line is added.
		24, 65, and others	The compile option -stack_protector/-stack_protector_all is added.
		25, 80, and others	The compile option -misra2012 is added.
		25	MISRA-C: 2012 rules are added to the descriptions of -ignore_files_misra and -check_language_extension.
		56	same_code is added to optimization items.
		82	A description regarding MISRA-C: 2012 rules is added to [Caution].
		83	-misra2012 is added.
		85	MISRA-C: 2012 rules are added.
		86	MISRA-C: 2012 rules are added.

Rev.	Date	Description	
		Page	Summary
1.02	Sep 14, 2015	149, 185, and others	The link option -SYmbol_forbid is added.
		156	[Restrictions] is added.
		174	[Syntax] and [Description] of -CRc are changed.
		186	[Syntax], [Description], and [Remark] of -Optimize are changed.
		236	The location of the description of -misra2004 is changed.
		237	The combination of -misra2004 and –misra2012 is added to the combinations of options with conflicting features.
		278	unsigned long long int is added to the types of integer constants.
		292, 293, and others	#pragma stack_protector and #pragma no_stack_protector are added.
		429	The contents of symbols that cannot be written as operands are changed.
		430	The description regarding operands is changed.
		431	The description regarding operands is changed.
		435	.EXITM and .EXITMA are deleted from macro directives.
		439	The maximum number of formal parameters and the description regarding a macro call are changed.
		440	[Description] is changed.
		441	The description on the case where there is no arrangement of statements (block) is deleted.
		442	[Description] is changed.
		443	[Description] is changed.
		444	[Description] is changed.
		451	[Description] is changed.
		469	"@$IMM_constant value" is added to assembler generated symbols.
		574	The following library functions are added. calloc, free, malloc, realloc
		636	The example of the startup routine is changed.
		651	[Action by User] is added to E0520014.
		651	[Action by User] is added to E0520020.
		675	[Action by User] is added to E0523005.
		695	[Action by User] is added to E0562200.
		696	[Action by User] is added to E0562201.
		697	[Action by User] is added to E0562320.

Rev.	Date	Description	
		Page	Summary
1.02	Sep 14, 2015	705	[Message] and [Explanation] are added to F0523029.
		705	[Explanation] of F0523030 is changed.
		716	[Action by User] is added to F0563113.
		722	M0523086 is added.
		725	[Action by User] is added to W0511179.
		757	The note regarding the K&R format is changed.
		761	The note regarding the separation operator is changed.
1.03	Jul 01, 2016	55, 56	pipeline is added to optimization items.
		73	[Detailed description] is changed.
		83, 84	The following MISRA-C:2012 rules are added. 2.6 2.7 9.2 9.3 12.1 12.3 12.4 14.4 15.1 15.2 15.3 15.4 15.5 15.6 15.7 16.1 16.2 16.3 16.4 16.5 16.6 16.7 17.1 17.7 18.4 18.5 19.2 20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12 20.13 20.14
		174	[Detailed description] is changed.
		210	[Remark] is changed.
		237	-Opipeline is added.
		334	[Restrictions] is changed.
		337	Embedded functions __set1, __clr1, and __not1 are added.
		476, 477	The contents of [V1.03 or later] are added.
		587	[Description] is changed.
		589	[Description] is changed.
		590	[Description] is changed.
		591	[Description] is changed.
		653, and others	Unnecessary messages are deleted.
		730	W0520171 is added.
1.04	Dec 01, 2016	13	The description of "License" is changed.
		13	"Standard and Professional Editions" is added.
		13	"Free Evaluation Editions" is added.
		20, 21	The description is changed.
		55	[Detailed description] is changed.
		58, 59	[Example of use] is changed.
		76	[Detailed description] is changed.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2016	86, 87	The following MISRA-C:2012 rules are added. 2.2 3.2 5.1 5.6 5.7 5.8 5.9 8.3 8.9 9.1 12.2 21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 21.10
		184	[Remark] is changed.
		186	[Detailed description] is changed.
		271	The description is changed.
		273, and others	The descriptions of the following implementation-defined items are changed. 4.1.3 (1), (4), (6), (7), (9), (12), (14), (16), (36), (37), (38), (39)
		285	The description is changed.
		298	The #pragma directives are added.
		332	[Restrictions] is changed.
		333, 334	[Example] is changed.
		336	[Restrictions] is changed.
		442	[Caution] is changed.
		443	[Caution] is changed.
		444	[Description] is changed.
		456	"Machine-Language Instruction Set" is added.
		594	[Description] is changed.
		596	[Description] is changed.
		597	[Description] is changed.
		599	[Description] is changed.
		609	[Caution] is added.
		626	[Caution] is added.
		634-642	The description is changed.
		644, 645	The description is changed.
		664	E0520020 is changed.
		666	E0520065 is changed.
		698	E0551406 is added.
		721	Unnecessary messages are deleted.
		728	F0563006 is added.
		729	F0563020 is changed.
		729	F0563115 is added.
		736	M0560700 is added.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2016	746	Unnecessary messages are deleted.
		753	W0561014 is added.
		772	"Controlling the Output of Bit Manipulation Instructions" is added.
		773	"Initialization of Stack Area at Startup" is added.
		781	The description is changed.
1.05	Jun 01, 2017	13	The description of "Standard and Professional Editions" is changed.
		25, 68, 77	The compile option -insert_nop_with_label is added.
		63	[Detailed description] is changed.
		72	[Detailed description] is changed.
		87	[Specification format] is changed.
		87, 88	The following MISRA-C:2012 rules are added. 12.5 13.2 13.5 17.5 17.8 21.13 21.15 21.16
		148	[Detailed description] is changed.
		149	[Detailed description] is changed.
		153, 164, 171	The link option -END_RECORD is added.
		160	[Detailed description] is changed.
		166	Table 2.9 is changed.
		179	[Example of use] is changed.
		187, 188	[Specification format] and [Detailed description] are changed.
		188	[Remark] is changed.
		191	[Detailed description] is changed.
		192	[Remark] is changed.
		207	[Detailed description] is changed.
		217	[Remark] is changed.
		224	[Remark] is added.
		250, 252, 253	The description of "Link map information" is changed.
		273, 274	The description of "Outputting the variable/function information file" is changed.
		274	The description of "How to use variable/function information file" is changed.
		284, 302, 303	The description of "#pragma directive" is changed.

Rev.	Date	Description	
		Page	Summary
1.05	Jun 01, 2017	303, 344, 345	"near/far function (#pragma near/#pragma far)" is added.
		303, 346	"Structure packing (#pragma pack/#pragma unpack)" is added.
		310	"Relationship with keywords and #pragma" is changed.
		336	[Restrictions] is changed.
		339, 340	[Restrictions] is changed.
		342	[Function] and [Restrictions] are changed.
		347	[Usage] is changed.
		435	[Syntax] and [Description] are changed.
		436	[Syntax] is changed.
		437	[Syntax] is changed.
		438	[Syntax] is changed.
		641-644	The following Interrupt Disabled Time are changed. acos, acosf, asin, asinf, atan, atanf, atan2, atan2f, tan, tanf, cosh, coshf, sinh, sinhf, tanh, tanhf, exp, expf, pow, powf, sqrt, sqrtf, scanf, sscanf, atof, _COM_atof_f, atoff, _COM_atoff_f, strtod, _COM_strtod_ff, strtof, _COM_strtof_ff
		647	The Interrupt Disabled Time of _COM_fdiv is changed.
		667, 668, 723, 732, 740, 744	The following messages are added. C0511200, C0519996, C0519997, C0550802, C0550804, C0550805, C0550806, C0550808, C0551800, C0564001, F0563103, W0511184, W0511185, W0523120, W0561015, W0561016
		670, 702, 715, 722, 743, 744	The following messages are changed. E0511200, E0551309, F0523073, F0563006, W0561004, W0561017
		744	Unnecessary messages are deleted.
		767	"Error output regarding an address not in the saddr access range" is added.
1.06	Dec 01, 2017	11,13	The C99 standard is supported.
		11	The section of "Limits" is deleted.
		24, 41, 42, 255, 298-304, 309, 331	The -lang compile option is added.

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2017	25, 64, 66, 255	The -change_message compile option is added.
		25, 71, 80-82, 255	The -control_flow_integrity compile option is added.
		26, 84, 85, 255, 256, 287, 289, 298-304, 309, 312, 324, 328, 330, 361	The -strict_std compile option is added.
		26, 105, 118, 255	The -unaligned_pointer_for_ca78k0r compile option is added.
		55	The contents of [Detailed description] of the -far_rom compile option are changed.
		57	The contents of [Detailed description] of the -O compile option are changed.
		65	The descriptions of [Specification format] and [Detailed description] of the -no_warning_num compile option are changed.
		92	The description of [Caution] of the -misra2004 compile option is changed.
		93, 94	The following MISRA-C:2012 rules are added.
		106, 107	The contents of [Detailed description] of the -convert_cc compile option are changed.
		115	The description of the operation of #pragma pack when the -convert_cc=iar compile option is specified is changed.
		155	The description of [Specification format] of the -warning assemble option is changed.
		156	The description of [Specification format] of the -no_warning assemble option is changed.
		160, 171, 173, 174, 181, 186	The -FIX_RECORD_LENGTH_AND_ALIGN link option is added.

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2017	160, 171, 173, 197	The -CFI link option is added.
		160, 171, 173, 198	The -CFI_ADD_Func link option is added.
		160, 171, 173, 199	The -CFI_IGNORE_Module link option is added.
		167	The description of [Restrictions] of the -Binary link option is changed.
		173, 185	The specifiable condition of the -BYte_count link option is changed.
		202, 203, 261, 267	CFI is added to the parameter of the -SHow link option.
		208	The description of [Caution] is added to the -Absolute_forbid link option.
		296	The description of "Predefined macro names" in "Implementation dependent items" is changed.
		303	The section of "Option to process in strict accordance with ANSI standard" is deleted.
		313-314	The section of "Macro" is added.
		334	The description of [Usage] of "Hardware interrupt handler (#pragma interrupt)" is changed.
		349	The description of control instructions and directives of the assembler is added to [Restrictions] in "Describing assembler instruction (#pragma inline_asm)".
		505	The following header files are added. inttypes.h, iso646.h, and stdbool.h
		654, 662	The description of the __control_flow_integrity function is added.
		709, 730, 747	The following messages are added. E0523087, F0563602, and W0561331
		719-721, 726-729, 732, 734, 744-747	The following messages are changed. E0562311, E0562340, E0562350, E0562351, E0562352, E0562353, E0562354, E0562355, E0562360, E0562361, E0562362, E0562363, E0562364, E0562365, E0562417, F0563004, F0563102, F0563123, F0563124, F0563431, M0560005, W0520062, W0561130, W0561184, W0561325, and W0561531

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2017	681	The following internal error messages are deleted. C0560901, C0560903, C0560904, C0560905, C0560906, C0560907, C0592xxx, C0592100, and C0592200
		682, 709, 716-721	The following error messages are deleted. E0511120, E0544003, E0544240, E0544854, E0560601, E0560602, E0560603, E0560604, E0560605, E0560606, E0560607, E0560608, E0560609, E0560610, E0560611, E0560612, E0560613, E0560614, E0560615, E0560616, E0560617, E0560618, E0560619, E0560620, E0560621, E0560622, E0560623, E0560624, E0560637, E0560638, E0560641, E0560660, E0562015, E0562017, E0562021, E0562112, E0562113, E0562143, E0562203, E0562222, E0562223, E0562323, E0562324, E0562331, E0562366, E0562400, E0562402, E0562403, E0562404, E0562405, E0562406, E0562407, E0562408, E0572000, E0572200, E0572500, E0572501, E0572502, E0573005, E0573007, E0573008, E0573009, E0573300, E0573303, E0573310, E0573320, E0592001, E0592002, E0592003, E0592004, E0592005, E0592006, E0592007, E0592008, E0592010, E0592013, E0592015, E0592016, E0592018, E0592019, E0592020, E0592101, E0592102, E0592201, E0593002, E0593003, E0593004, E0594000, E0594001, and E0594002
		724, 726, 728-730	The following fatal error messages are deleted. F0542001, F0542002, F0544302, F0544802, F0560001, F0560002, F0560003, F0560004, F0560005, F0560006, F0560007, F0560008, F0560009, F0560010, F0560011, F0560012, F0560013, F0560101, F0560102, F0560103, F0560104, F0560105, F0560106, F0560107, F0560108, F0560109, F0560110, F0560112, F0560113, F0560114, F0560115, F0560201, F0560202, F0560203, F0560204, F0560208, F0560209, F0560210, F0560213, F0560215, F0560216, F0560217, F0560218, F0560219, F0560220, F0560301, F0560302, F0560303, F0560304, F0560306, F0560307, F0560309, F0560310, F0560311, F0560404, F0560405, F0560407, F0560409, F0560411, F0560414, F0560415, F0560417, F0560419, F0560421, F0560423, F0560424, F0560502, F0560503, F0560627, F0560629, F0560630, F0560631, F0560633, F0560634, F0560635, F0560636, F0560649, F0560650, F0560652, F0560657, F0560658, F0560661, F0560662, F0560701, F0560705, F0560707, F0560708, F0560712, F0561001, F0561002, F0561003, F0561004, F0561005, F0561006, F0561007, F0561008, F0561009, F0561010, F0561011, F0561012, F0561013, F0561014, F0561015, F0561016, F0561019, F0562001, F0562002, F0562003, F0562004, F0562005, F0562006, F0562007, F0562008, F0562009, F0562014, F0562028, F0563120, F0563311, F0563312, F0563313, F0563400, F0563420, F0578200, F0578201, F0578202, F0578203, F0578204, F0578205, F0578206, F0578207, F0578208, F0578209, F0578210, F0578212, F0578213, F0578214, F0578215, F0578216, F0578217, F0578218, F0578219, F0578220, F0578221, F0593113, F0593114, F0595001, F0595002, F0595003, and F0595004
		732	The following information messages are deleted. M0560001, M0560002, M0560102, M0560103, M0560300, M0560510, M0560511, M0560512, M0560600, M0592150, M0592151, M0592152, M0592153, M0592154, M0592155, M0592156, M0592157, M0592250, M0592251, M0592252, M0592253, M0592270, M0592280, M0592281, M0592282, M0594201, M0594202, and M0594203

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2017	742, 744-747	The following warning messages are deleted. W0542101, W0544001, W0544002, W0560111, W0560116, W0560205, W0560206, W0560207, W0560212, W0560214, W0560305, W0560308, W0560312, W0560313, W0560314, W0560315, W0560316, W0560401, W0560402, W0560403, W0560406, W0560408, W0560410, W0560412, W0560413, W0560416, W0560418, W0560420, W0560422, W0560501, W0560625, W0560628, W0560639, W0560640, W0560642, W0560643, W0560644, W0560645, W0560647, W0560651, W0560653, W0560654, W0560655, W0560656, W0560659, W0560702, W0560706, W0560709, W0560710, W0560711, W0561110, W0561180, W0561190, W0561192, W0561194, W0561304, W0561321, W0561327, W0561420, W0561430, W0561500, W0561501, W0561502, W0561510, W0562010, W0562011, W0562012, W0562013, W0562015, W0562016, W0562017, W0562018, W0562019, W0562020, W0562021, W0562022, W0562023, W0562024, W0562025, W0562026, W0562027, W0571600, W0578306, W0578307, W0578308, W0578309, W0578310, W0578311, W0578315, W0578322, W0592009, W0592011, W0592012, W0592017, W0592103, W0592104, W0592105, W0594100, W0594101, W0594102, W0594103, W0594104, W0594105, W0594106, W0594107, W0594110, W0594111, W0594112, W0594113, W0594114, W0594115, W0594116, W0594117, W0594118, W0594119, W0594124, W0594125, W0594126, W0594127, W0594128, W0594129, W0594130, W0594131, W0594132, W0594133, W0594134, W0594135, W0594140, W0594150, W0594151, W0594160, W0594161, W0594162, W0594163, W0594164, W0594165, and W0594166
		751	The description of "Controlling the Output of Bit Manipulation Instructions" is changed.
		757	The description of "Allocating to sections accessible with short instructions" is changed.
1.07	Jun 01, 2018	19	The description of "Specification format" is changed.
		42	The descriptions of [Remark] is added to the -lang compile option.
		52	The descriptions of [Detailed description] of the -preprocess compile option is changed.
		65	The descriptions of [Specification format] of the -no_warning_num compile option is changed.
		66	The descriptions of [Specification format] of the -change_message compile option is changed.
		95	The descriptions of [Detailed description] of the -ignore_files_misra compile option is changed.
		96	The descriptions of [Detailed description] of the -check_language_extension compile option is changed.
		155	The descriptions of [Specification format] of the -warning assemble option is changed.
		156	The descriptions of [Specification format] of the -no_warning assemble option is changed.
		160, 171, 194	The -SPLIT_VECT link option is added.

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	162, 227, 232	The -CHECK_OUTPUT_ROM_AREA link option is added.
		164	The descriptions of [Specification format] of the -Input link option is changed.
		167	The descriptions of [Specification format] of the -Binary link option is changed.
		169	The descriptions of [Specification format] of the -DEFine link option is changed.
		172-174	The descriptions of [Specification format], [Detailed description], and [Remark] of the -FOrm link option are changed.
		177	The descriptions of [Specification format] of the -RECord link option is changed.
		178	The descriptions of [Specification format] of the -END_RECORD link option is changed.
		180	The descriptions of [Specification format], [Detailed description], and [Remark] of the -OUtput link option are changed.
		183	The descriptions of [Specification format] of the -NOMessage link option is changed.
		188, 189, 191	The descriptions of [Specification format], [Detailed description], and [Remark] of the -CRc link option are changed.
		193	The descriptions of [Specification format] and [Detailed description] of the -VECTN link option are changed.
		198	The descriptions of [Specification format] of the -CFI_ADD_Func link option is changed.
		199	The descriptions of [Specification format], [Detailed description], and [Example of use] of the -CFI_IGNORE_Module link option are changed.
		211	The descriptions of [Specification format] of the -STARt link option is changed.
		228	The descriptions of [Specification format] of the -CPu link option is changed.
		243	The descriptions of [Specification format] and [Remark] of the -REName link option are changed.
		244	The descriptions of [Specification format] of the -DELete link option is changed.
		245	The descriptions of [Specification format] of the -REPlace link option is changed.
		248	The descriptions of [Specification format] of the -CHange_message link option is changed.
		324	The descriptions of [Function] of "Specifying memory allocation area (__near /__far)" is changed.
		342	The descriptions of [Caution] of "Changing compiler output section name (#pragma section)" is changed.
		353	The descriptions of [Restrictions] of "Absolute address allocation specification (#pragma address)" is changed.
		449	The descriptions of [Description] of the .DB directive is changed.
		450	The descriptions of [Description] of the .DB2 directive is changed.
		499	Table 6.1 is changed.

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	504	The following libraries are added. rl78nm4s99.lib, rl78cm4s99.lib, rl78em4s99.lib, and rl78em8s99.lib
		504, 505	The description of "Rule for Naming Libraries" is changed.
		507	The description of "Library Function" is changed.
		508	The descriptions of [Description] of the assert function is changed.
		509, 513	The isblank function is added.
		526-530	The section of "Functions for greatest-width integer types" is added.
		584, 593-595	The snprintf function is added.
		584, 607-609	The vsnprintf function is added.
		584	The descriptions of "Outline" of sprintf and vsprintf are changed.
		585-588	The descriptions of [Syntax] and [Description] of the printf function are changed. [Restrictions] is added.
		590, 592	The descriptions of [Description] of the scanf function is changed. [Restrictions] is added.
		596-599	The descriptions of outline, [Syntax], and [Description] of the sprintf function are changed. [Restrictions] is added.
		601, 603	The descriptions of [Description] of the sscanf function is changed. [Restrictions] is added.
		604-606	The descriptions of [Syntax] and [Description] of the vprintf function are changed. [Restrictions] is added.
		610-612	The descriptions of outline, [Syntax], and [Description] of the vsprintf function are changed. [Restrictions] is added.
		618, 623	The atoll function is added.
		618, 626	The strtold function is added.
		618, 628	The strtoll function is added.
		618, 630	The strtoull function is added.
		618, 647	The llabs function is added.
		618, 648	The lldiv function is added.

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	624	The descriptions of [Syntax] of the strtod function is changed. [Restrictions] is added.
		625	The descriptions of [Syntax] of the strtof function is changed. [Restrictions] is added.
		627	The descriptions of [Syntax] of the strtol function is changed.
		629	The descriptions of [Syntax] of the strtoul function is changed.
		643	The descriptions of [Return value] of the abs function is changed.
		645	The descriptions of [Return value] of the labs function is changed.
		650	The descriptions of [Syntax] of the memcpy function is changed.
		652	The descriptions of [Syntax] of the strcpy function is changed.
		653	The descriptions of [Syntax] of the strncpy function is changed.
		654	The descriptions of [Syntax] of the strcat function is changed.
		655	The descriptions of [Syntax] of the strncat function is changed.
		666	The descriptions of [Syntax] of the strtok function is changed.
		675-679	The following functions are added. isblank, imaxabs, imaxdiv, strtoimax, _COM_strtoimax_ff, strtoumax, _COM_strtoumax_ff, snprintf, vsnprintf, atoll, _COM_atoll_f, strtold, _COM_strtold_ff, strtoll, _COM_strtoll_ff, strtoull, _COM_strtoull_ff, llabs, and lldiv
		703, 706-710, 713, 714, 728, 737-741, 744	The following messages are added. E0511177, E0511182, E0520069, E0520117, E0520175, E0520223, E0520655, E0520744, E0520747, E0523003, E0523014, E0523038, E0523044, E0523048, E0523077, E0523078, F0520571, F0523088, W0511181, W0511183, W0520159, W0520221, W0520222, W0520240, W0520606, W0520609, W0520819, W0520966, W0520967, W0520968, W0521037, W0521039, W0521040, W0523018, W0523085, and W0561143
		709-711, 713, 723, 739, 740, 744	The following messages are changed. E0520643, E0520757, E0520977, E0521051, E0521052, E0521649, E0562135, E0562142, W0520257, W0520940, W0521051, W0561183, and W0561184
		701	C0510000 is deleted.

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	703, 704, 706-710	The following error messages are deleted. E0511111, E0511112, E0511118, E0511119, E0511122, E0511125, E0511126, E0511127, E0511132, E0511136, E0511137, E0511138, E0511139, E0511140, E0511141, E0511142, E0511148, E0511155, E0511157, E0511158, E0511159, E0511160, E0511161, E0511167, E0511173, E0511175, E0511176, E0511200, E0512001, E0520002, E0520079, E0520096, E0520103, E0520123, E0520126, E0520131, E0520133, E0520135, E0520150, E0520153, E0520157, E0520160, E0520194, E0520195, E0520196, E0520220, E0520227, E0520230, E0520239, E0520241, E0520242, E0520243, E0520244, E0520245, E0520246, E0520248, E0520249, E0520250, E0520251, E0520252, E0520255, E0520257, E0520258, E0520259, E0520262, E0520263, E0520264, E0520265, E0520266, E0520269, E0520276, E0520277, E0520278, E0520279, E0520280, E0520281, E0520282, E0520283, E0520285, E0520286, E0520287, E0520288, E0520289, E0520290, E0520291, E0520292, E0520293, E0520294, E0520297, E0520298, E0520299, E0520300, E0520302, E0520304, E0520305, E0520306, E0520307, E0520308, E0520309, E0520310, E0520311, E0520312, E0520314, E0520315, E0520316, E0520317, E0520318, E0520319, E0520320, E0520321, E0520322, E0520323, E0520326, E0520327, E0520328, E0520329, E0520330, E0520332, E0520333, E0520334, E0520335, E0520336, E0520337, E0520338, E0520339, E0520341, E0520342, E0520343, E0520344, E0520345, E0520346, E0520347, E0520348, E0520349, E0520350, E0520351, E0520352, E0520353, E0520354, E0520356, E0520357, E0520358, E0520359, E0520360, E0520363, E0520364, E0520365, E0520366, E0520367, E0520369, E0520371, E0520372, E0520373, E0520378, E0520380, E0520384, E0520386, E0520389, E0520390, E0520391, E0520392, E0520394, E0520397, E0520403, E0520405, E0520407, E0520408, E0520410, E0520412, E0520413, E0520415, E0520416, E0520417, E0520418, E0520424, E0520427, E0520429, E0520432, E0520433, E0520434, E0520435, E0520436, E0520437, E0520438, E0520439, E0520440, E0520441, E0520442, E0520443, E0520449, E0520452, E0520456, E0520457, E0520458, E0520459, E0520461, E0520463, E0520464, E0520466, E0520467, E0520468, E0520470, E0520471, E0520473, E0520475, E0520476, E0520477, E0520478, E0520481, E0520484, E0520485, E0520486, E0520487, E0520489, E0520493, E0520496, E0520498, E0520500, E0520501, E0520502, E0520503, E0520504, E0520505, E0520506, E0520507, E0520508, E0520510, E0520511, E0520515, E0520516, E0520517, E0520518, E0520519, E0520529, E0520530, E0520531, E0520532, E0520536, E0520540, E0520543, E0520544, E0520546, E0520548, E0520551, E0520555, E0520556, E0520558, E0520559, E0520598, E0520599, E0520601, E0520603, E0520604, E0520605, E0520612, E0520615, E0520616, E0520620, E0520647, E0520651, E0520656, E0520658, E0520661, E0520663, E0520664, E0520665, E0520666, E0520667, E0520668, E0520669, E0520670, E0520673, E0520674, E0520691, E0520692, E0520693, E0520694, E0520695, E0520696, E0520697, E0520698, E0520701, E0520703, E0520704, E0520706, E0520707, E0520709, E0520710, E0520711, E0520717, E0520718, E0520719, E0520724, E0520725, E0520726, E0520727, E0520728, E0520730, E0520734, E0520735, E0520742, E0520750, E0520751, E0520752, E0520753, E0520754, E0520755, E0520756, E0520758, E0520759, E0520769, E0520771, E0520772, E0520773, E0520774, E0520775, E0520776, E0520779, E0520782, E0520785, E0520786, E0520787, E0520788, E0520789, E0520790, E0520791, E0520792, E0520795, E0520799, E0520800, E0520801, E0520803, E0520804, E0520805, E0520807, E0520808, E0520809, E0520810, E0520812, E0520817, E0520818, and E0520819

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	710-714	The following error messages are deleted. E0520822, E0520824, E0520827, E0520828, E0520832, E0520834, E0520835, E0520840, E0520841, E0520842, E0520844, E0520847, E0520848, E0520849, E0520850, E0520851, E0520854, E0520855, E0520857, E0520864, E0520865, E0520868, E0520871, E0520872, E0520873, E0520875, E0520876, E0520877, E0520878, E0520879, E0520880, E0520881, E0520882, E0520885, E0520890, E0520891, E0520892, E0520893, E0520894, E0520896, E0520898, E0520901, E0520915, E0520916, E0520928, E0520929, E0520930, E0520934, E0520937, E0520939, E0520948, E0520952, E0520954, E0520955, E0520956, E0520960, E0520963, E0520964, E0520971, E0520972, E0520975, E0520978, E0520979, E0520980, E0520982, E0520985, E0520987, E0520988, E0520989, E0520990, E0520994, E0520995, E0520996, E0520998, E0520999, E0521001, E0521006, E0521007, E0521009, E0521010, E0521011, E0521013, E0521014, E0521015, E0521017, E0521018, E0521019, E0521020, E0521021, E0521022, E0521023, E0521032, E0521034, E0521035, E0521042, E0521043, E0521044, E0521047, E0521054, E0521061, E0521065, E0521066, E0521067, E0521075, E0521076, E0521081, E0521086, E0521087, E0521088, E0521089, E0521146, E0521161, E0521163, E0521201, E0521204, E0521212, E0521227, E0521229, E0521230, E0521254, E0521255, E0521280, E0521282, E0521291, E0521292, E0521295, E0521303, E0521304, E0521311, E0521312, E0521315, E0521317, E0521318, E0521320, E0521321, E0521322, E0521323, E0521324, E0521325, E0521326, E0521327, E0521344, E0521345, E0521349, E0521350, E0521351, E0521355, E0521365, E0521372, E0521380, E0521382, E0521398, E0521403, E0521404, E0521405, E0521424, E0521425, E0521436, E0521437, E0521441, E0521442, E0521445, E0521534, E0521535, E0521536, E0521542, E0521543, E0521557, E0521558, E0521574, E0521576, E0521577, E0521583, E0521586, E0521587, E0521588, E0521589, E0521590, E0521593, E0521596, E0521597, E0521598, E0521602, E0521619, E0521620, E0521624, E0521629, E0521631, E0521632, E0521634, E0521645, E0521646, E0523042, E0523057, E0523058, E0523059, E0523066, E0523069, E0523070, E0523071, and E0523072
		728	The following fatal error messages are deleted. F0511128, F0512003, F0520016, F0520190, F0520219, F0520542, F0520583, F0520584, F0520641, F0520869, F0520919, F0520926, F0521083, F0521151, F0521335, F0521336, F0521337, F0521338, F0523054, F0523055, F0523056, F0523073, F0523300, F0523301, and F0523302
		736	The following information messages are deleted. M0520009, M0520018, M0520111, M0520128, M0520174, M0520193, M0520237, M0520261, M0520324, M0520381, M0520399, M0520400, M0520479, M0520487, M0520534, M0520535, M0520549, M0520618, M0520652, M0520678, M0520679, M0520815, M0520831, M0520863, M0520866, M0520949, M0521348, M0521353, M0521380, M0521381, M0523009, and M0523033,

Rev.	Date	Description	
		Page	Summary
1.07	Jun 01, 2018	737-741	The following warning messages are deleted. W0511143, W0511144, W0511156, W0511166, W0511168, W0511169, W0511170, W0511171, W0511172, W0511179, W0519999, W0520001, W0520014, W0520019, W0520045, W0520046, W0520047, W0520054, W0520066, W0520085, W0520086, W0520101, W0520118, W0520139, W0520144, W0520147, W0520157, W0520171, W0520181, W0520185, W0520224, W0520225, W0520226, W0520232, W0520262, W0520280, W0520284, W0520296, W0520300, W0520326, W0520335, W0520368, W0520370, W0520377, W0520381, W0520382, W0520414, W0520430, W0520497, W0520512, W0520514, W0520522, W0520523, W0520533, W0520541, W0520552, W0520553, W0520554, W0520611, W0520614, W0520617, W0520650, W0520657, W0520662, W0520691, W0520692, W0520708, W0520720, W0520722, W0520723, W0520737, W0520748, W0520760, W0520780, W0520783, W0520794, W0520802, W0520806, W0520812, W0520825, W0520826, W0520829, W0520830, W0520831, W0520836, W0520837, W0520912, W0520925, W0520936, W0520941, W0520942, W0520948, W0520959, W0520961, W0520962, W0520970, W0520973, W0520984, W0520991, W0520997, W0521028, W0521030, W0521050, W0521055, W0521105, W0521145, W0521163, W0521192, W0521193, W0521194, W0521197, W0521211, W0521213, W0521218, W0531235, W0521273, W0521285, W0521290, W0521294, W0521296, W0521301, W0521302, W0521307, W0521308, W0521310, W0521316, W0521319, W0521342, W0521346, W0521361, W0521373, W0521374, W0521375, W0521386, W0521396, W0521400, W0521420, W0521427, W0521433, W0521443, W0521444, W0521546, W0521547, W0521548, W0521551, W0521553, W0521561, W0521564, W0521565, W0521566, W0521570, W0521607, W0521611, W0521632, W0521635, W0521636, W0521651, W0523042, W0523060, W0523063, W0523064, and W0523120
		752	The section of "Specifying standard library functions when C99 standard is specified by an individual option" is added.
		755	The section of "Version of Compiler Package" is added.
1.08	Dec 01, 2018	14	The description of "Optimizing linker" is changed.
		26, 90, 97, 259	The -misra_intermodule compile option is added.
		63	[Detailed description] of the -goptimize compile option is changed.
		75	[Detailed description] of the -switch compile option is changed.
		93	The following MISRA-C:2012 rules are added. 8.5 8.6
		137	[Detailed description] of the -goptimize assemble option is changed.
		163, 239, 246	The -LIB_RENName link option is added.
		171	[Remark] of the -ENTry link option is changed.
		174	Table 2.9 is changed.
		203	[Detailed description] of the -SHow link option is changed.
		204	[Remark] of the -SHow link option is changed.
		244	[Remark] of the -MEMory link option is changed.

Rev.	Date	Description	
		Page	Summary
1.08	Dec 01, 2018	245	[Detailed description] and [Remark] of the -REName link option are changed.
		254	[Remark] of the -Total_size link option is changed.
		291-307	The configuration of "Basic Language Specifications" was reviewed.
		315-322	The configuration of "Extended Language Specifications" was reviewed.
		538, 541, 732	The fpclassify function is added.
		538, 542, 732	The isfinite function is added.
		538, 543, 732	The isinf function is added.
		538, 544, 732	The isnan function is added.
		538, 545, 732	The isnormal function is added.
		538, 546, 732	The signbit function is added.
		538, 549, 732	The acosl function is added.
		538, 552, 732	The asinl function is added.
		538, 555, 732	The atanl function is added.
		538, 558, 732	The atan2l function is added.
		538, 561, 732	The cosl function is added.
		538, 564, 732	The sinl function is added.
		538, 567, 733	The tanl function is added.

Rev.	Date	Description	
		Page	Summary
1.08	Dec 01, 2018	538, 568, 733	The acosh function is added.
		538, 569, 733	The acoshf function is added.
		538, 570, 733	The acoshl function is added.
		538, 571, 733	The asinh function is added.
		539, 572, 733	The asinhf function is added.
		539, 573, 733	The asinhl function is added.
		539, 574, 733	The atanh function is added.
		539, 575, 733	The atanhf function is added.
		539, 576, 733	The atanhl function is added.
		539, 579, 733	The coshl function is added.
		539, 582, 733	The sinhl function is added.
		539, 585, 733	The tanhl function is added.
		539, 588, 733	The expl function is added.
		539, 591, 733	The frexpl function is added.
		539, 594, 733	The ldexpl function is added.
		539, 597, 733	The logl function is added.

Rev.	Date	Description	
		Page	Summary
1.08	Dec 01, 2018	539, 600, 734	The log10l function is added.
		539, 601, 734	The log1p function is added.
		539, 602, 734	The log1pf function is added.
		539, 603, 734	The log1pl function is added.
		539, 606, 734	The modfl function is added.
		540, 609, 734	The fabsl function is added.
		540, 612, 734	The powl function is added.
		540, 615, 734	The sqrtl function is added.
		540, 618, 734	The ceill function is added.
		540, 621, 734	The floorl function is added.
		540, 624, 734	The fmodl function is added.
		553	[Description] of the atan function is changed.
		554	[Description] of the atanf function is changed.
		584	[Description] of the tanhf function is changed.
		632, 655-658, 735	The vscanf function is added.
		632, 665-668, 735	The vsscanf function is added.
		637-640	[Syntax], [Description] and [Restrictions] of the scanf function are changed.

Rev.	Date	Description	
		Page	Summary
1.08	Dec 01, 2018	648-651	[Syntax], [Description] and [Restrictions] of the sscanf function are changed.
		734	"Interrupt Disabled Time" of the sqrt function is changed.
		734	"Interrupt Disabled Time" of the sqrtf function is changed.
		738	"Use of .data" and "Use of .bss" of the strtok function are changed.
		743	The descriptions in "Stack area allocation, stack pointer setting, and stack area initialization" is changed.
		749, 750	"ROMization" is deleted and "Creating ROM Images" is added.
		759, 764, 785, 796	The following messages are added. C0520000, C0529000, E0520079, E0562600, W0520070
		782, 801	The following messages are changed. E0562320, W0561101
		785	E0562500 is deleted.
		809	The descriptions in "Initialization of Stack Area at Startup" is changed.
		815, 816	The descriptions in "Method for allocating variables or constants to each area" is changed.
		819	The descriptions in "Executing a program in RAM" is changed.
1.09	Nov 01, 2019	11	The description of "Copyrights" is changed.
		87	"Interpretation when omitted" of the -nest_comment compile option is changed.
		92	The following MISRA-C:2012 rules are added. 8.13 14.2 14.3
		159, 163, 171, 174, 244	The -ALLOW_DUPLICATE_MODULE_NAME link option is added.
		291-295	The description of "Implementation-defined behavior of C90" is changed.
		300, 301	(71) and (72) are changed.
		348, 349	The descriptions in "Changing compiler output section name (#pragma section)" is changed.

Rev.	Date	Description	
		Page	Summary
1.09	Nov 01, 2019	540-542, 609-614, 630-653, 657-677, 681, 684, 788-790	The following functions are added. scalbn, scalbnf, scalbnl, scalbln, scalblnf, scalblnl, nearbyint, nearbyintf, nearbyintl, rint, rintf, rintl, lrint, lrintf, lrintl, llrint, llrintf, llrintl, round, roundf, roundl, lround, lroundf, lroundl, llround, llroundf, llroundl, trunc, truncf, truncl, copysign, copysignf, copysignl, nan, nanf, nanl, fdim, fdimf, fdiml, fmax, fmaxf, fmaxl, fmin, fminf, fminl, isgreater, isgreaterequal, isless, islessequal, islessgreater, isunordered, va_copy

Rev.	Date	Description	
		Page	Summary
1.10	Nov 01, 2020	Front cover	The target CPU cores are added.
		16	The tool usage information file is added to Table 2.1.
		23, 50	The description and [Detailed description] for the -preinclude compile option are changed.
		24, 71, 79, 80	The -stuff compile option is added.
		58, 59	The following optimization items are added: branch_chaining, align
		72	[Specification format] and [Detailed description] of the -dbl_size compile option is changed.
		166, 243, 259	The -VERBOSE link option is added.
		220	[Detailed description] for the -USER_OPT_BYTE link option is changed.
		264	The following options are added to the section "Specifying multiple times of options": -Obranch_chaining, -Oalign
		328	The format specification for the #pragma directive in Table 4.15 is changed.
		330, 331	[Usage], [Restrictions], and [Remark] in "Using saddr area (__saddr)" are changed.
		352	[Usage] in "Changing compiler output section name" is changed.
		375, 376 and others	Wording "embedded function" is changed to "intrinsic function".
		375, 376	The "Function" descriptions for the following intrinsic functions in Table 4.16 are changed: __mulu, __mului, __mulsi, __mulul, __mulsl
		375-377	The format of the function declarations in the Table 4.16 is changed to ANSI-C.
		442, 444	[Specification format] and [Detailed description] of the .SECTION directive are changed.
		821, 836, 846-849, 857, 860, 868, 869	The following messages are added: C0580013, C0580014, C0580015, C0580016, C0580900, C0580901, C0580902, C0580903, C0580904, C0580905, C0580906, C0580907, E0550271, E0580001, E0580002, E0580003, E0580004, E0580005, E0580006, E0580007, E0580008, E0580009, E0580010, E0580011, E0580012, E0580017, E0580100, E0580104, E0580105, E0580106, E0580200, E0580201, E0580202, E0580203, E0580204, E0580300, E0580301, E0580302, E0580303, E0580305, E0580307, E0580308, E0580309, E0580310, E0580311, E0580312, E0580313, E0580314, E0580315, E0580316, E0580317, E0580318, F0580101, F0580102, F0580103, F0580399, W0520171, W0580304, W0580306

Rev.	Date	Description	
		Page	Summary
1.11	Nov 01, 2021	11	The description of "GENERAL" is changed.
		18	The description of "Specification format" in "Command line operation" is changed.
		49	[Detailed description] of the -I compile option is changed.
		57	The descriptions of the inline_size optimization item is changed.
		144	[Detailed description] of the -include assemble option is changed.
		211	[Detailed description] of the -Optimize link option is changed.
		223	[Detailed description] of the -SECURITY_ID link option is changed.
		329	The description of "#pragma directive" is changed.
		381, 385	$ is added to symbols and the description of "Conventions of symbol description" is changed.
		470, 474	.WEAK directive is added.
		475, 481	.ALIAS directive is added.
		475, 482	.TYPE directive is added.
		821	An error in the example is corrected in "Reference of Argument Defined by Other Language".
		825, 860, 862, 868	The following messages are changed: E0511178, F0563430, W0511180, W0511185, W0561016, W0561017
		829, 865	Errors in the following messages are corrected: E0520137, W0521053
		839	The following message is added: E0550272
		876	Errors are corrected in "Controlling the Output of Bit Manipulation Instructions [V1.04 or later]".

Rev.	Date	Description	
		Page	Summary
1.12	Dec 01, 2022	11	The description in "Copyrights" is changed.
		15	The C++ source file is added in Table 2.1.
		18	The description in "Command line operation" is changed.
		23, 39	The description of the -use_mda compile option and its information in [Specification format] and [Detailed description] are changed, and [Caution] is added.
		23, 41	The description of the -lang compile option and its information in [Specification format], [Detailed description], and [Remark] are changed.
		49	[Detailed description] for the -I compile option is changed.
		56, 57, 59, 60	[Specification format], [Detailed description] and [Example of use] for the -O compile option are changed.
		141	[Detailed description] for the -define assemble option is changed.
		163, 176, 206, 207	The -RAM_INIT_TABLE_SECTION link option is added.
		164, 219, 226	The -SECURITY_OPT_BYTE link option is added.
		164, 219, 229, 230	The -FLASH_SECURITY_ID link option is added.
		164, 219, 232-234	The -SPLIT_SECTION link option is added.
		164, 219, 235-237	The -STRIDE_DSP_MEMORY_AREA link option is added.
		165, 219, 246	The -DSP_MEMORY_AREA link option is added.
		194, 196	[Detailed description] and [Example of use] for the -CRc link option are changed.
		214	[Detailed description] for the -Optimize link option is changed.
		225	[Detailed description] for the -OCDBG link option is changed.

Rev.	Date	Description	
		Page	Summary
1.12	Dec 01, 2022	227	[Detailed description] for the -SECURITY_ID ink option is changed.
		238	[Detailed description] for the -DEBUG_MONITOR link option is changed.
		278, 279	Information about specification of multiple options is changed.
		287	FLASH_SECUR_ID is added as a relocation attribute.
		355	The description in "Comparison with pointers" in "Specifying memory allocation area (__near /__far)" is changed.
		456, 458, 459	[Syntax] and [Description] for the .SECTION directive are changed.
		461	[Description] for the .CSEG directive is changed.
		529	The following section is added in Table 6.1. .flash_security_id
		824-826	"Initialization of RAM area sections by using an initialization table [V1.12 or later]" is added.
		828-831	Table 8.1 is changed.
		871, 886	The following messages are changed: F0520571 and W0561017
		880, 886	The following messages are added: W0511186, W0511187, and W0561018
1.13	Dec 01, 2023	11	The description of "Copyrights" is changed.
		13-15, and others	The description of the library generator is added.
		18, 19	The description of "Command line operation" is changed.
		21, 280-290	"Library generator options [V1.13.00 or later]" is added.
		22, 37, 40	The -use_mach compile option is added.
		22, 44	The description of the -P compile option and its [Detailed description] and [Example of use] are changed.
		24, 87	The classification of compile options is changed.
		100	[Remark] is added to the -misra_intermodule compile option.
		165, 214, 220	The -ALLOW_OPTIMIZE_ENTRY_BLOCK link option is added.

Rev.	Date	Description	
		Page	Summary
1.13	Dec 01, 2023	185	[Detailed description] of the -Rom link option is changed.
		193	[Detailed description] of the -PADDING link option is changed.
		195	[Detailed description] of the -CRc link option is changed.
		238	[Example of use] of the -STRIDE_DSP_MEMORY_AREA link option is changed.
		240	[Example of use] of the -DEBUG_MONITOR link option is changed.
		241	[Detailed description] and [Remark] of the -RRM link option are changed.
		371	[Caution] in "Specifying inline function (__inline)" is changed.
		373, 374	[Usage] in "Hardware interrupt handler (#pragma interrupt)" is changed.
		377	[Usage] in "Software interrupt handler (#pragma interrupt_brk)" is changed.
		403	In Table 4.16, the "Function" column for the following intrinsic functions is changed: __mulul and __mulsl
		542	The "Description" column in Table 6.1 is changed.
		543	Note of Table 6.1 is deleted.
		546	The -rrm option is added in Table 6.2.
		547	The title of Chapter 7 is changed.
		549	In Table 7.2, the "Outline" column for inttypes.h is changed.
		715	[Caution] is added in "setjmp".
		847, 848	The contents of "General registers and ES/CS registers whose values are guaranteed" are changed.
		848	"System registers whose values are guaranteed" is deleted.
		850	The description of "Return value" is changed.
		852	The description of "Register" in "Calling of C Language Routine from Assembly Language" is changed.
		854	The following message is changed: C0519996
		855, 875, 891, 892, 904	The following messages are added: C0590001, E0562114, F0593000, F0593021, F0593300, F0593302, F0593303, F0593305, F0593320, F0593321, F0593322, F0593324, F0593325, F0593326, F0593327, F0593328, F0593329, F0593330, W0591300, and W0591301

CC-RL User's Manual

Publication Date: Rev.1.00 Aug 01, 2014
Rev.1.13 Dec 01, 2023

Published by: Renesas Electronics Corporation

CC-RL

Renesas Electronics Corporation

R20UT3123EJ0113