User's Manual

CC-RH

Compiler

User's Manual

Applicable Revision

V1.00.00 to V2.06.00

Target Device

RH850 Family

Target CPU Cores:

G3M, G3K, G3MH, G3KH, G4MH

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.11 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.

© 2023 Renesas Electronics Corporation. All rights reserved.

How to Use This Manual

This manual describes the role of the CC-RH compiler for developing applications and systems for RH850 family, and provides an outline of its features.

Readers	This manual is intended for users who wish to understand the functions of the CC-RH and design software and hardware application systems.
Purpose	This manual is intended to give users an understanding of the functions of the CC-RH 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 FUNCTIONAL 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 10
1.1 Outline 10
1.2 Special Features 10
1.3 Copyrights 10
1.4 License 10
1.5 Standard and Professional Editions 10
1.6 Free Evaluation Editions 11
2. COMMAND REFERENCE 12
2.1 Overview 12
2.2 I/O Files 13
2.3 Environment Variable 15
2.4 Method for Manipulating 16
2.4.1 Command line operation 16
2.4.2 Subcommand file usage 18
2.5 Option 19
2.5.1 Compile options 20
2.5.2 Assemble options 132
2.5.3 Link options 170
2.6 Specifying Multiple Options 249
2.6.1 Priority 249
2.6.2 Incompatible features 249
2.6.3 Dependencies 249
2.6.4 Relationship with #pragma directives 250
3. OUTPUT FILES 251
3.1 Assemble List File 251
3.1.1 Structure of the assemble list 251
3.1.2 Assemble list 251
3.1.3 Section list 252
3.1.4 Command line information 252
3.2 Link Map File 253
3.2.1 Structure of link map 253
3.2.2 Option information 253
3.2.3 Error information 254
3.2.4 Link map information 254
3.2.5 Total section size 255
3.2.6 Symbol information 255

3.2.7 Contents of the Function List . . . 258
3.2.8 Cross reference information . . . 258
3.2.9 CRC information . . . 259
3.3 Link Map File (When Objects Are Combined) . . . 260
3.3.1 Structure of link map . . . 260
3.3.2 Header information . . . 260
3.3.3 Option information . . . 260
3.3.4 Error information . . . 261
3.3.5 Entry information . . . 261
3.3.6 Combined address information . . . 261
3.3.7 Address overlap information . . . 262
3.4 Library List File . . . 263
3.4.1 Structure of the library list . . . 263
3.4.2 Option information . . . 263
3.4.3 Error information . . . 264
3.4.4 Library information . . . 264
3.4.5 Module, section, and symbol information within the library . . . 265
3.5 Intel HEX File . . . 266
3.5.1 Structure of the Intel HEX file . . . 266
3.5.2 Start linear address record . . . 267
3.5.3 Expanded linear address record . . . 267
3.5.4 Start address record . . . 268
3.5.5 Expanded address record . . . 268
3.5.6 Data record . . . 269
3.5.7 End record . . . 269
3.6 Motorola S-record File . . . 271
3.6.1 Structure of the Motorola S-record file . . . 271
3.6.2 S0 record . . . 272
3.6.3 S1 record . . . 272
3.6.4 S2 record . . . 273
3.6.5 S3 record . . . 273
3.6.6 S7 record . . . 273
3.6.7 S8 record . . . 274
3.6.8 S9 record . . . 274
4. COMPILER LANGUAGE SPECIFICATIONS . . . 275
4.1 Basic Language Specifications . . . 275
4.1.1 Implementation-defined behavior of C90 . . . 275
4.1.2 Implementation-defined behavior of C99 . . . 283
4.1.3 Internal representation and value area of data . . . 298
4.1.4 Register mode . . . 305
4.2 Extended Language Specifications . . . 307

4.2.1 Reserved words . . . 307
4.2.2 Macro . . . 307
4.2.3 C99 language specifications supported in conjunction with C90 . . . 308
4.2.4 Compiler generated symbols . . . 309
4.2.5 #pragma directive . . . 310
4.2.6 Using extended language specifications . . . 311
4.2.6.1 Allocation of function and data to section . . . 312
4.2.6.2 Describing assembler instruction . . . 319
4.2.6.3 Inline expansion . . . 321
4.2.6.4 Controlling interrupt level . . . 324
4.2.6.5 Interrupt/Exception processing handler . . . 325
4.2.6.6 Disabling or enabling maskable interrupts . . . 333
4.2.6.7 Intrinsic functions . . . 335
4.2.6.8 Structure type packing . . . 337
4.2.6.9 Bit field assignment . . . 346
4.2.6.10 Core number specification (for a multi-core device) . . . 348
4.2.6.11 Specifying alignment value for branch destination addresses . . . 351
4.2.6.12 Detection of stack smashing [Professional Edition only] . . . 352
4.2.6.13 Half-precision floating-point type [Professional Edition only] [V1.05.00 or later] . . . 354
4.2.6.14 Detection of writing to control registers or insertion of synchronization processing [Professional Edition only] [V1.06.00 or later]356
4.2.7 Modification of C source . . . 359
5. ASSEMBLY LANGUAGE SPECIFICATIONS . . . 360
5.1 Description of Source . . . 360
5.1.1 Description . . . 360
5.1.2 Expressions and operators . . . 365
5.1.3 Arithmetic operators . . . 367
5.1.4 Logic operators . . . 375
5.1.5 Relational operators . . . 380
5.1.6 Shift operators . . . 389
5.1.7 Byte separation operators . . . 392
5.1.8 2-byte separation operators . . . 395
5.1.9 Section operators . . . 399
5.1.10 Other operator . . . 402
5.1.11 Restrictions on operations . . . 404
5.1.12 Identifiers . . . 405
5.2 Directives . . . 406
5.2.1 Outline . . . 406
5.2.2 Section definition directives . . . 407
5.2.3 Symbol definition directives . . . 416
5.2.4 Compiler output directives . . . 419
5.2.5 Data definition/Area reservation directives . . . 426

5.2.6 External definition/External reference directives. . . . 436
5.2.7 Macro directives . . . 440
5.3 Control Instructions . . . 449
5.3.1 Outline . . . 449
5.3.2 Assembler control instructions . . . 450
5.3.3 File input control instructions . . . 458
5.3.4 Conditional assembly control instructions. . . . 461
5.4 Macro . . . 470
5.4.1 Outline . . . 470
5.4.2 Usage of macro . . . 470
5.4.3 Macro operator . . . 470
5.5 Reserved Words . . . 471
5.6 Predefined Macro Names . . . 472
5.7 Assembler Generated Symbols . . . 472
5.8 Instruction Set . . . 473
5.9 Extension of Assembly Language . . . 481

6. SECTION SPECIFICATIONS . . . 525

6.1 Sections . . . 525
6.1.1 Section concatenation . . . 525
6.2 Special Symbol . . . 527

7. LIBRARY FUNCTIONAL SPECIFICATIONS . . . 529

7.1 Supplied Libraries . . . 529
7.2 Header Files. . . . 532
7.3 Reentrancy. . . . 533
7.4 Library Function . . . 533
7.4.1 Program diagnostic functions . . . 533
7.4.2 Functions with variable arguments . . . 535
7.4.3 Character string functions. . . . 539
7.4.4 Memory management functions . . . 555
7.4.5 Character conversion functions . . . 561
7.4.6 Character classification functions . . . 564
7.4.7 Standard I/O functions . . . 577
7.4.8 Standard utility functions. . . . 611
7.4.9 Non-local jump functions . . . 635
7.4.10 Mathematical functions. . . . 638
7.4.11 RAM section initialization function . . . 667
7.4.12 Peripheral device initialization function. . . . 670
7.4.13 Operation runtime functions . . . 672
7.4.14 Checks for indirect function calls function. . . . 675
7.4.15 Dynamic memory management functions . . . 676
7.5 Usage of Data Sections and List of Reentrancy . . . 682

8. STARTUP 691
8.1 Outline 691
8.2 Startup Routine 691
8.2.1 Initialization routine for hardware 691
8.2.2 Initialization routines of user programs 694
8.2.3 Passing information between projects 697
8.3 Coding Example 698
8.4 Symbols 704
8.4.1 __gp_data 704
8.4.2 __ep_data 705
8.4.3 __pc_data 705
8.5 Creating ROM Images 706
8.6 PIC/PID Facility 707
8.6.1 PIC 707
8.6.2 PIROD 707
8.6.3 PID 707
8.6.4 Referencing from a position-independent program to a position-dependent program 707
8.6.5 Restrictions on PIC/PID facility 709
8.6.6 Startup routine 710
9. FUNCTION CALL INTERFACE SPECIFICATIONS 718
9.1 Function Call Interface 718
9.1.1 General-purpose registers guaranteed before and after function calls 718
9.1.2 Setting and referencing arguments and return values 718
9.1.3 Address indicating stack pointer 720
9.1.4 Stack frame 721
9.2 Calling of Assembly Language Routine from C Language 723
9.3 Calling of C Language Routine from Assembly Language 724
9.4 Reference of Argument Defined by Other Language 725
9.5 General-purpose Registers 725
10. MESSAGE 726
10.1 General 726
10.2 Message Formats 726
10.3 Message Types 726
10.4 Messages 726
10.4.1 Internal errors 727
10.4.2 Errors 729
10.4.3 Fatal errors 750
10.4.4 Information 756
10.4.5 Warnings 757
11. CAUTIONS 767

11.1 Volatile Qualifier . . . 767
11.2 -Xcpu Option Specification for Assembler . . . 768
11.3 Controlling the Output of Bit Manipulation Instructions [V1.05.00 or later] . . . 768
11.4 Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler . . . 769
11.5 Version of Compiler Package . . . 769

A. QUICK GUIDE . . . 770

A.1 Variables (C Language) . . . 770
A.1.1 Allocating to sections accessible with short instructions . . . 770
A.1.1.1 GP relative access . . . 770
A.1.1.2 EP relative access . . . 771
A.1.2 Changing allocated section . . . 772
A.1.2.1 Changing the area to be allocated using the #pragma section directive . . . 772
A.1.2.2 Changing the area to be allocated using the -Xsection option . . . 773
A.1.2.3 Change the allocated area using the -Xpreinclude option . . . 773
A.1.3 Defining variables for use during standard and interrupt processing . . . 774
A.1.4 Defining const constant pointer . . . 775
A.2 Functions . . . 776
A.2.1 Changing area to be allocated to . . . 776
A.2.2 Calling away function . . . 776
A.2.3 Embedding assembler instructions . . . 776
A.2.4 Executing a specific routine from RAM . . . 777
A.3 Variables (Assembler) . . . 777
A.3.1 Defining variables with no initial values . . . 777
A.3.2 Defining variable with initial values . . . 778
A.3.3 Defining const constants . . . 778

Revision Record . . . C - 1

1. GENERAL

This document is the user's manual for the RH850 family's C compiler CC-RH V1.00 to V2.06.
This chapter provides a general outline of CC-RH.

1.1 Outline

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

1.2 Special Features

CC-RH is equipped with the following special features.

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

(2) Advanced optimization
Advanced optimization is used, applying global program optimization as well as conventional optimization.
This yields smaller, faster code, and also reduces build times.

(3) High portability
The program supports porting programs from the existing SuperH RISC engine C/C++ compiler.
In addition, the industry-standard DWARF2 and DWARF3 format is used for debugging information.

(4) Multifunctional
Static analysis and other functionality is provided via linking between CS+.

1.3 Copyrights

This software uses LLVM and Protocol Buffers.

- LLVM is copyright of University of Illinois at Urbana-Champaign.
- Protocol Buffers is copyright of Google Inc.

Other software components are copyright of 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 V1.05 and later versions of CC-RH.
Use a license for V2 or later versions of CC-RH in development for the RH850 G4 core.

1.5 Standard and Professional Editions

There are two editions of the compilers, the Standard and the Professional editions.
The Standard editions support an C90 and C99 standards C-language specification, 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 4.10, List of Supported #pragma Directive.

For the libraries supported only in the Professional Edition, see "7.1 Supplied Libraries".

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 a restriction becomes applicable to the sizes produced by linkage.

- The restriction on the section sizes which can be allocated to the ROM area is up to 256 Kbytes in total. A linker error occurs when the size exceeds 256 Kbytes.

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 commercial edition.

Examples are given below.

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

We do not supply the following services for the evaluation editions.Consider purchasing a commercial 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 the build tool (CC-RH).

2.1 Overview

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

CC-RH consists of the following commands. A single driver (ccrh) controls all phases from compilation to linking.

ccrh: Compilation driver start command
asrh: Assembler start command
rlink: Optimizing linker start command

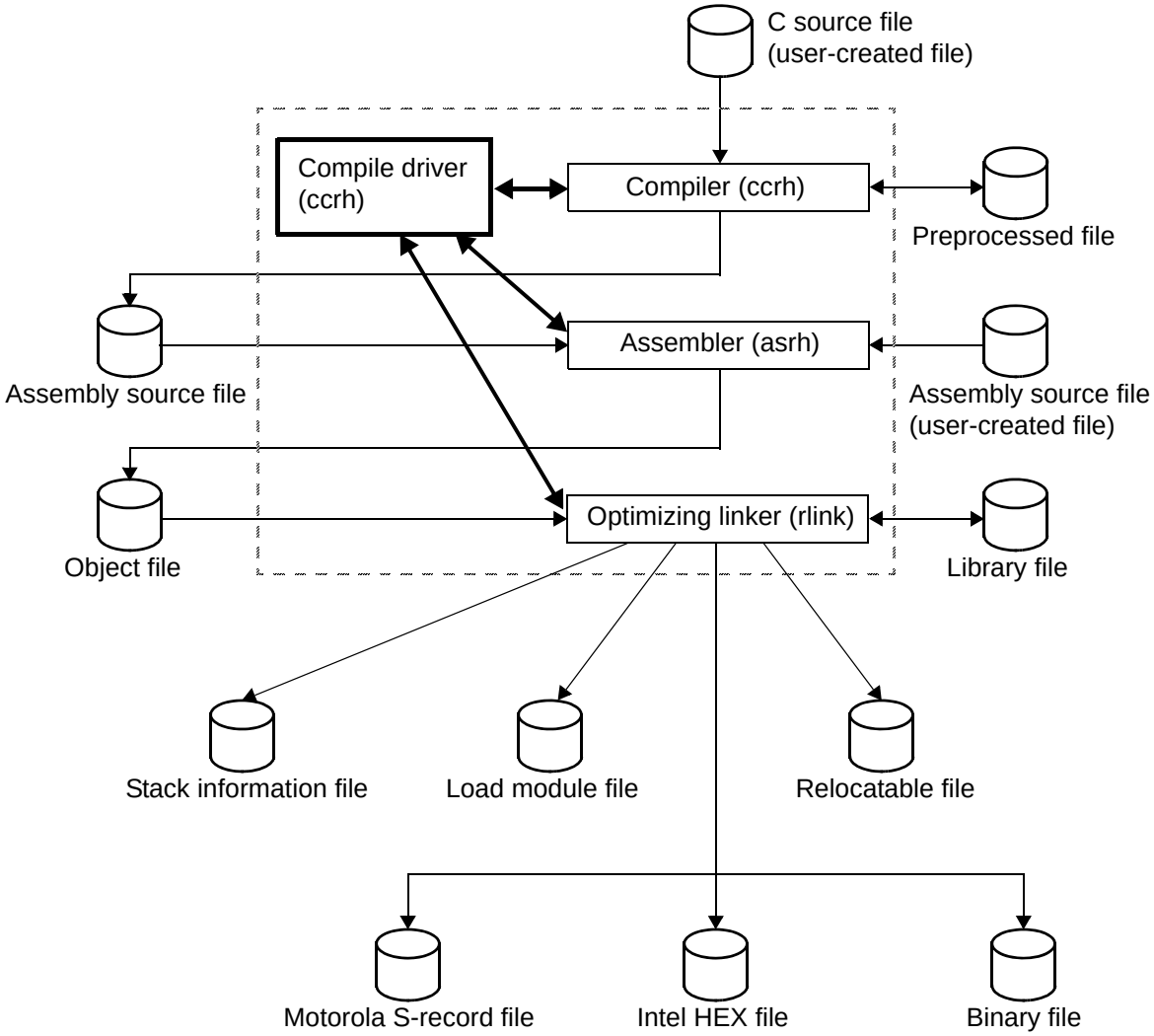
Processing of each command is shown below.

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

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

(3) Optimizing linker (rlink)
Links an object file and library file, and then generates an executable object file (load module file) on the target system.
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, and conversion to Intel HEX files and Motorola S-record files.

Figure 2.1 Operation Flow of ccrh



2.2 I/O Files

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

Table 2.1 I/O Files of ccrh Command

File Type	Extension	I/O	Description
C source file	.c	I	Source file described in C language This is created by the user.
Preprocessed file	.i[Note 1]	O	File which the execution result of preprocess processing for the input file is output This is an ASCII image file. 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.

File Type	Extension	I/O	Description
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 -Xasm_option=-Xprn_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 -Xlk_option=-form=library option is specified.
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 -Xlk_option=-form=absolute option is specified. If you specify the -Xlk_option option but not the -form option, the command assumes that the above option has been specified.
Relocatable file	.rel[Note 1]	O	Relocatable object file This file is output when the -Xlk_option=-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 -Xlk_option=-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 -Xlk_option=-form=stype option is specified.
Binary file	.bin[Note 1]	O	Load module file converted into the binary format This file is output when the -Xlk_option=-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 -Xlk_option=-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 -Xlk_option=-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 -Xlk_option=-list option is specified.
Stack information file	.sni	O	List file which has information of the stack capacity This file is output when the -Xlk_option=-stack option is specified.
External symbol allocation information file	.bls[Note 1]	O	External variable allocation information file used by the compiler in optimizing access to external variables This file is output when the -Omap option is specified.
Static analysis information file	free	I/O	Information file which this product uses The extension is free, but ".cref" is recommended. This file is output when the -Xcref option is specified.

File Type	Extension	I/O	Description
Error message file	free	O	File which contains error messages The extension is free, but ".err" is recommended. This file is output when the -Xerror_file option is specified.
Subcommand file	free	I	File which contains the parameters of the execution program This is created by the user.
The exclusive control check setting file	free	I	This is the file input from CS+.
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

You can launch the ccrh command (the compilation driver) to perform compilation, assembly, linking, and other actions. The assembler (asrh) and optimizing linker (rlink) can also start by itself.

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

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

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

```
>rlink[{Δfile|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 Assemble 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.
  However, "@" cannot be used at the beginning of a file name because it is regarded as the subcommand file specification.
  "-" 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 file is 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 ccrh, and then assembly source file "file1.asm" is generated.
Next, assembly source file "file1.asm" and "file2.asm" are assembled by asrh, and then object file "file1.obj" and "file2.obj" are generated.
In addition, the assemble list file is 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.

```
>ccrh file1.c file2.asm file3.obj -Xasm_option=-Xprn_path -Xlk_option=-list -
osample.abs -Xcommon=rh850
```

Remark In the ccrh command line, use the -Xasm_option option to specify an option dedicated to asrh; to specify an option dedicated to rlink, use the -Xlk_option option.

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

```
>ccrh -c file1.c file2.asm -Xasm_option=-Xprn_path -Xcommon=rh850
```

Remark In the ccrh command line, use the -Xasm_option option to specify an option dedicated to asrh.

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 ccrh, and then assembly source file "file1.asm" is generated.

```
>ccrh -S file1.c -Xcommon=rh850
```

Assembly source file "file1.asm" and "file2.asm" are assembled by asrh, and then object file "file1.obj" and "file2.obj" are generated.
In addition, the assemble list file is output to the current folder.

```
>asrh file1.asm -Xprn_path -Xcommon=rh850
>asrh file2.asm -Xprn_path -Xcommon=rh850
```

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
```

2.4.2 Subcommand file usage

A subcommand file is a file that options and file names specified for a 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 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 -Xcharacter_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.
  These special characters themselves are not included in the command line of the ccrh command and deleted.

" (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.

```
-Xcommon=rh850
-c
-Dtest
-Idir
-Osize
```

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

```
>ccrh @sub.txt -ofile.obj file.c
```

The command line is expanded as follows.

```
>ccrh -Xcommon=rh850 -c -Dtest -Idir -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.
- 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. [V1.04.00 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. [V1.05.00 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 ccrh options for each phase.
Compile phase -> See "2.5.1 Compile options"
Assemble phase -> See "2.5.2 Assemble options"
Link phase -> See "2.5.3 Link options"

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.

Table 2.2 Compile Options

Classification	Option	Description
Version/help display specification	-V	This option displays the version information of ccrh.
	-h	This option displays the descriptions of ccrh options.
Output file specification	-o	This option specifies the output file name.
	-Xobj_path	This option specifies the folder to save an object file generated during compilation.
	-Xasm_path	This option specifies the folder to save an assembly source file generated during compilation.
	-Xprep_path	This option specifies the folder to save the preprocessed file.
Source debugging control	-g	This option outputs information for source debugging.
	-g_line [V1.05.00 or later]	This option enhances information for source debugging at optimization.
Device specification	-Xcommon	This option specifies that an object file common to the various devices is generated.
	-Xcpu	This option specifies that an object for the specified core is generated.
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.
Preprocessor control	-D	This option defines preprocessor macros and assembler symbols.
	-U	This option deletes the definition of the preprocessor macro or assembler symbol by the -D option.
	-I	This option specifies the folder to search include files.
	-Xpreinclude	This option specifies the file that is included at the top of the compilation unit.
	-Xpreprocess	This option controls outputting the result of preprocessing.

Classification	Option	Description
C language control	-lang [V1.07.00 or later]	This option specifies the language standard.
	-strict_std [V1.07.00 or later]	This option processes the C source program in strict accordance with the language standard.
	-Xenum_type	This option specifies which integer type the enumeration type handles.
	-Xvolatile	This option specifies external variables as volatile.
	-Xcheck	This option checks the compatibility of a C source file.
	-Xmisra2004 [Professional Edition only]	This option checks source code against the MISRA-C: 2004 rules.
	-Xmisra2012 [Professional Edition only]	This option checks source code against the MISRA-C: 2012 rules.
	-Xignore_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.
	-Xcheck_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] [V2.01.00 or later]	This option checks source code in multiple files against the MISRA-C:2012 rules.
	-Xuse_fp16 [Professional Edition only] [V1.05.00 or later]	This option selects the use of the half-precision floating-point type.
Japanese/Chinese character control	-Xcharacter_set	This option specifies the Japanese/Chinese character code.
Optimization specification	-O	This option specifies the optimization level or the details of each optimization items.
	-Xintermodule	This option performs inter-module optimization.
	-Xinline_strcpy	This option performs inline expansion of standard library functions "strcpy", "strcmp", "memcpy", and "memset" calls.
	-Xmerge_string	This option merges string literals.
	-Xalias	This option performs optimization with consideration for the type of the data indicated by the pointer.
	-Xmerge_files	This option merges two or more C source files and compiles them.
	-Xwhole_program	This option performs optimization assuming that the files to be compiled comprise the entire program.
	-library [V2.00.00 or later]	This option performs inline expansion for calling the standard library functions.
	-goptimize [V2.01.00 or later]	This option generates information for link-time optimization.

Classification	Option	Description
Generated code control	-Xpack	This option performs the structure packing.
	-misalign [V2.04.00 or later]	This option generates an instruction string that performs a misaligned memory access.
	-Xbit_order	This option specifies the order of bit-field members.
	-Xpass_source	This option outputs a C source program as a comment to the assembly source file.
	-Xswitch	This option specifies a mode in which the code of a switch statement is to be output.
	-Xreg_mode	This option specifies the register mode.
	-Xreserve_r2	This option reserves the r2 register.
	-r4 [V1.07.00 or later]	This option specifies how to handle the r4 register.
	-Xep	This option specifies how to handle the ep register.
	-Xfloat	This option controls generating floating-point calculation instructions.
	-Xfxu [V2.00.00 or later]	This option controls usage of FXU instructions.
	-Xcall_jump	This option controls generating function-call branch instructions.
	-Xfar_jump	This option controls outputting far jump.
	-Xdiv	This option generates the div and divu instructions for division.
	-Xcheck_div_ov	This option checks the OV flag at division.
	-relaxed_math [V2.00.00 or later]	This option generates a floating-point calculation code with efficiency given precedence over strictness.
	-Xuse_fmaf	This option generates product-sum operation instructions.
	-use_recipf [V2.00.00 or later]	This option generates the recipf instructions.
	-approximate [V2.02.00 or later]	This option replaces floating-point calculations with equivalent approximate calculations.
	-Xunordered_cmpf	This option detects invalid operation exceptions in floating-point comparison.
	-Xmulti_level	This option specifies the generation of a multi-core program.
	-Xpatch	This option applies a patch.
	-Xdbl_size	This option specifies the data size of double and long double type.
	-Xround	This option specifies the mode for rounding floating-point constants.
	-Xalign4	This option specifies the alignment value for branch destination addresses.
	-Xstack_protector/ -Xstack_protector_all [Professional Edition only]	This option generates a code for detection of stack smashing.
	-Xsection	This option specifies the default sections for data.

Classification	Option	Description
Generated code control	-stuff [V2.03.00 or later]	This option allocates variables to sections separated according to the number of alignment.
	-Xcheck_exclusion_control [V1.04.00 or later]	This option enables checking of exclusive control.
	-Xresbank_mode [V2.00.00 or later]	This option specifies the operating mode of the resbank instruction.
	-insert_dbtag_with_label [V1.06.00 or later]	This option controls insertion of the dbtag instruction.
	-store_reg [Professional Edition only] [V1.06.00 or later]	This option controls detection of writing to control registers or insertion of synchronization processing between registers.
	-control_flow_integrity [Professional Edition only] [V1.07.00 or later]	This option generates code for the detection of illegal indirect function calls.
	-pic [V1.07.00 or later]	This option enables the PIC facility.
	-pirod [V1.07.00 or later]	This option enables the PIROD facility.
	-pid [V1.07.00 or later]	This option enables the PID facility.
Information file output control	-Xcref	This option outputs the static analysis information file.
Error output control	-Xerror_file	This option outputs error messages to a file.
Warning message output control	-Xno_warning	This option suppresses outputting warning messages of the specified number.
	-change_message [V1.07.00 or later]	This option changes specified warning messages into error messages.
Phase individual option specification	-Xasm_option	This option specifies assemble options.
	-Xlk_option	This option specifies link options.
Subcommand file specification	@	This option specifies a subcommand file.

Version/help display specification

The version/help display specification options are as follows.

- -V
- -h

-V

This option displays the version information of ccrh.

[Specification format]

```
-V
```

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

[Detailed description]

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

[Example of use]

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

```
>ccrh -V -Xcommon=rh850
```

-h

This option displays the descriptions of ccrh options.

[Specification format]

```
-h
```

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

[Detailed description]

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

[Example of use]

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

```
>ccrh -h -Xcommon=rh850
```

Output file specification

The output file specification options are as follows.

- -o
- -Xobj_path
- -Xasm_path
- -Xprep_path

-o

This option specifies the output file name.

[Specification format]

```
-ofile
```

- Interpretation when omitted
  The output file name differs depending on the specificated option.
  The file is output to the current folder.
  - 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 a load module file name has been specified as *file*.
- 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:

```
>ccrh -osample.abs -Xcommon=rh850 main.c
```

-Xobj_path

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

[Specification format]

```
-Xobj_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".
- 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:

```
>ccrh -Xobj_path=D:\sample -Xcommon=rh850 main.c
```

-Xasm_path

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

[Specification format]

```
-Xasm_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".
- 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:

```
>ccrh -Xasm_path=D:\sample -Xcommon=rh850 main.c
```

-Xprep_path

This option specifies the folder to save the preprocessed file.

[Specification format]

```
-Xprep_path[=path]
```

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

[Detailed description]

- This option specifies the folder to save a preprocessed file as *path*.
- If an existing folder is specified as *path*, the preprocessed 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 file is output, it will be saved with *path* as the file name.
  If two or more preprocessed files are output, an error will occur.
  An error will occur if a nonexistent file is specified.
- 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 file is only saved for the last source file to be specified.
- 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 file is only saved for the last source file to be specified.

[Example of use]

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

```
>ccrh -Xprep_path=D:\sample -Xcommon=rh850 main.c
```

Source debugging control

The source debugging control option is as follows.

- -g
- -g_line [V1.05.00 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.
- If this option and an optimization option are specified at the same time, the ease of debugging could be affected.

[Example of use]

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

```
>ccrh -g -Xcommon=rh850 main.c
```

-g_line [V1.05.00 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:

```
>ccrh -g -g_line main.c
```

Device specification

The device specification options are as follows.

- -Xcommon
- -Xcpu

-Xcommon

This option specifies that an object file common to the various devices is generated.

[Specification format]

```
-Xcommon=series
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies that an object file common to the various devices is generated.
- This option is invalid in V2.00.00 or later versions. If this option is specified, it will be ignored but no error will occur due to the compatibility with conventional versions. No warning is output in this case.
- v850e3v5 or rh850 can be specified for *series*.
- An error will occur in any of the following cases.
  - When *series* is omitted
  - When a parameter that is not specifiable is specified for *series*
  - When this option is omitted [V1.01.00 or earlier]

[Remark]

This option does not affect the output code.
When selecting the instruction set to be used, specify the -Xcpu option.

-Xcpu

This option specifies that an object for the specified core is generated.

[Specification format]

```
-Xcpu=core
```

- Interpretation when omitted
  An object for G3M is generated.

[Detailed description]

- This option specifies that an object for core *core* is generated.
- The items that can be specified as *core* are shown below.

g3m	Generates an object for G3M.
g3k	Generates an object for G3K.
g3mh	Generates an object for G3MH. [V1.02.00 or later]
g3kh	Generates an object for G3KH. [V1.03.00 or later]
g4mh	Generates an object for G4MH. [V2.00.00 or later]

- If this option is specified more than once, the last specification is valid.
- An error will occur in any of the following cases.
  - When the parameter is omitted
  - When a parameter that is not specifiable is specified

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 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 -Xpreprocess option.

[Example of use]

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

```
>ccrh -P -Xcommon=rh850 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:

```
>ccrh -S -Xcommon=rh850 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:

```
>ccrh -c -Xcommon=rh850 main.c
```

Preprocessor control

The preprocessor control options are as follows.

- -D
- -U
- -I
- -Xpreinclude
- -Xpreprocess

-D

This option defines preprocessor macros and assembler symbols.

[Specification format]

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

- Interpretation when omitted
  None

[Detailed description]

- This option defines *name* as a preprocessor macro or user-defined symbol of the assembler.
- This is equivalent to adding "#define *name def*" or "SET *name def*" (only assembly source program) at the beginning of the source program.
- If *name* contains characters that are allowed in an assembler symbol, but which cannot be used in a preprocessor macro ("@", ".", and "~"), a warning will be output, and it is defined as an assembler symbol only.
- This option can be used to redefine C language macros that have been defined already: __LINE__, __FILE__, __DATE__, __TIME__, and __CCRH__ (except for -D__CCRH__[=1]).
  An error will occur if these are redefined when the input file is a C source file.
- 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 and assembler symbol, the option specified last will be valid.

[Example of use]

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

```
>ccrh -Dsample=256 -Xcommon=rh850 main.c
```

-U

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

[Specification format]

```
-Uname[,name]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes the definition of the preprocessor macro or user-defined symbol of the assembler *name* by the -D option.
- This is equivalent to adding "#undef *name*" at the beginning of the source program.
- An error will occur if *name* is omitted.
- This option cannot delete the definition by describing "#define *name def*" and ".SET *name def*" (only assembly source program).
- This option can be used to undefine C language macros that have been defined already, but it cannot undefine the following macros: __LINE__, __FILE__, __DATE__, __TIME__, __CCRH__, or __CCRH.
  An error will occur if these are specified for *name*.
- This option can be specified more than once.
- If both this option and -D option are specified for the same preprocessor macro and assembler symbol, the option specified last will be valid.

[Example of use]

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

```
>ccrh -Utest -Xcommon=rh850 main.c
```

-I

This option specifies the folder to search include files.

[Specification format]

```
-Ipath[,path]...
```

- Interpretation when omitted
  The include file is searched from the standard include file folder.

[Detailed description]

- This option specifies the folder to search include files which are read by preprocessor directive "#include" or assembler control instruction "$INCLUDE/$BINCLUDE" as *path*.
  Include files are searched according to the following sequence.

(1) #include

(a) Folder with source files (When files are specified by using " ")

(b) 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).)

(c) Standard include file folder

(2) $INCLUDE/$BINCLUDE

(a) 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).)

(b) Folder with source file

(c) 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 include files from the current folder, folder D:\include, the standard folder in that order, describe as:

```
>ccrh -ID:\include -Xcommon=rh850 main.c
```

-Xpreinclude

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

[Specification format]

```
-Xpreinclude=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*.
- 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:

```
>ccrh main.c -Xpreinclude=sample.h -Xcommon=rh850
```

-Xpreprocess

This option controls outputting the result of preprocessing.

[Specification format]

```
-Xpreprocess=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, 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[Note].

<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* " (e.g. "\\%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:

```
>ccrh -Xpreprocess=comment,line -P -Xcommon=rh850 main.c
```

The following example is equivalent to the example above.

```
>ccrh -Xpreprocess=comment -Xpreprocess=line -P -Xcommon=rh850 main.c
```

C language control

The C language control options are as follows.

- -lang [V1.07.00 or later]
- -strict_std [V1.07.00 or later]
- -Xenum_type
- -Xvolatile
- -Xcheck
- -Xmisra2004 [Professional Edition only]
- -Xmisra2012 [Professional Edition only]
- -Xignore_files_misra [Professional Edition only]
- -Xcheck_language_extension [Professional Edition only]
- -misra_intermodule [Professional Edition only] [V2.01.00 or later]
- -Xuse_fp16 [Professional Edition only] [V1.05.00 or later]

-lang [V1.07.00 or later]

This option specifies the language standard.

[Specification format]

```
-lang={c|c99}
```

- Interpretation when omitted
  Compilation is performed according to the C90 standard.

[Detailed description]

- This option specifies the language standard of the C source file.
- If the -lang=c option is specified or this option is omitted, compilation is performed according to the C90 standard.
- If the -lang=c99 option is specified, compilation is performed according to the C99 standard.
- If a value other than c or c99 is specified, an error will occur.

[Remark]

- This compiler does not support a part of language standards.
  - Some 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

-strict_std [V1.07.00 or later]

This option processes the C source program in strict accordance with the language standard.

[Specification format]

```
-strict_std [V1.07.00 or later]
-Xansi [Compatible use with V1.06.00 and earlier versions]
```

- Interpretation when omitted
  Compatibility with the conventional C language specifications is conferred and processing continues after warning is output. Even if -lang=c99 option is not specified, some of the specifications added by C99 are accepted.

[Detailed description]

- This option processes the C source program in strict accordance with the language standard specified by the -lang option and outputs an error or warning for a specification that violates the standard.
- For the predefined macros that are valid when this option is specified or not specified, see "4.2.2 Macro".
- Processing when compiling in strict adherence to the language standard is as follows.
  - When conforming to C90
    - Bit fields
      An error will occur if a type other than an int, signed int, or unsigned int type is specified in a bit field.
      If this option is not specified, specifying a type other than an int type will be enabled (A warning will not be output).
    - *#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*".
    - Argument of function for which #pragma inline is specified
      If the type of the return value or parameter is different but type conversion is possible between the specified function call and definition, an error will occur.
      If this option is not specified, the type of the return value is converted to the type at the call side, the parameters are converted to the type of the function definition, and inline expansion is performed.
    - Basic type
      An error will occur if a _Bool, long long, unsigned long long, or __fp16 type is specified.
    - Structure and union specifiers
      If the member declaration list does not include named members, then an error message will be output indicating that the list has no effect.
  - When conforming to C99
    - *#line-number*
      An error will occur.
      When this option is not specified, it is treated in the same manner as "#line *line-number*".
    - Parameters of functions declared with #pragma inline
      If the type of a return value or parameter is different but type conversion is possible between the specified function call and definition, then an error will occur.
      When this option is not specified, the type of the return value is converted to the type at the calling site, the parameters are converted to the type of the function definition, and inline expansion is performed.
    - Basic type
      An error will occur if a __fp16 type is specified.
    - Structure and union specifiers
      If the member declaration list does not include named members, then an error will occur.

-Xenum_type

This option specifies which integer type the enumeration type handles.

[Specification format]

```
-Xenum_type=string
```

- Interpretation when omitted
  The enumeration type is handled as signed int.

[Detailed description]

- This option specifies which integer type the enumeration type handles.
- The items that can be specified as *string* are shown below.
  An error will occur if any other item is specified.

auto	Each enumerated type is handled as the smallest integer type capable of expressing all the enumerators in that type.

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

[Example of use]

- To handle each enumerated type as the smallest integer type capable of expressing all the enumerators in that type, describe as:

```
>ccrh -Xenum_type=auto -Xcommon=rh850 main.c
```

-Xvolatile

This option specifies external variables as volatile.

[Specification format]

```
-Xvolatile
```

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

[Detailed description]

- This option handles all external variables as if they were volatile-declared.
  The access count and access order for external variables are exactly the same as those described in the C source file.

[Example of use]

- To handle all external variables as if they were volatile-declared, describe as:

```
>ccrh -Xvolatile -Xcommon=rh850 main.c
```

-Xcheck

This option checks the compatibility of a C source file.

[Specification format]

```
-Xcheck=comp
```

- Interpretation when omitted
  The compatibility of a C source file is not checked.

[Detailed description]

- This option checks the C source file coded for the compiler specified as *comp*. It checks for option specifications and source code that will impact compatibility when compiled with this compiler, and outputs warnings or errors about any impacts found.
- The items that can be specified as *comp* are shown below.
  An error will occur if any other item is specified.

shc	Checks the C source file that has been coded for the SH compiler.

- An error will occur if *comp* is omitted.
- The main check items are shown below.
  - Options: -Xbit_order=*pos*
    The settings which are not defined in the language specification and depend on implementation differ in each compiler.
    Confirm the selections of the options which were output in the message.
  - Extended functions: #pragma section, #pragma entry#pragma stacksize, #pragma address, #pragma global_register
    There is a possibility that extended specifications will affect program operation.
    Confirm the descriptions on the extended specifications which were output in the message.
  - volatile qualified variables
    The sizes of reads and writes may differ between compilers.
    This compiler may access bit fields with a volatile decorator as a smaller size than the declared type, but the SH compiler will access them as the size of the declared type.
  - Integer promotion of binary operations
    The result of binary operation (such as addition, subtraction, multiplication, division, or comparison) using unsigned int-type and long-type operands may differ from that obtained by using the SH compiler.
    The SH compiler calculates this operation in signed long if the -strict_ansi option is not specified.
    This compiler calculates this operation after converting the operands to the unsigned int type.
  - Types of integer constants exceeding type signed long
    The SH compiler makes values in the range that can be expressed as type unsigned long into type signed long long.
    This compiler makes values in the range that can be expressed as type unsigned long into type unsigned long.
  - Bit field allocation
    The SH compiler does not allocate bits to contiguous areas when the type of a bit field differs from that of the previous bit field.
    This compiler may allocate bits to contiguous areas according to the -Xpack option setting.
- No message will be output for structure and bit field member allocation.
  See "4.1.3 Internal representation and value area of data" about declarations that take assignment into account.

[Example of use]

- To check the C source file that has been coded for the SH compiler, describe as:

```
>ccrh -Xcheck=shc -Xcommon=rh850 main.c
```

-Xmisra2004 [Professional Edition only]

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

[Specification format]

```
-Xmisra2004=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*.
- This option cannot be specified simultaneously with the -Xmisra2012 option.
- If this option is specified simultaneously with the -lang=c99 option, this option is ignored. At this time, a warning will be output.
- 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 15.3 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.
- The __fp16 type is handled as the float type during the check. For the effects of this handling, see the description of the -Xcheck_language_extension [Professional Edition only] option.

[Example of use]

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

```
>ccrh -Xmisra2004=apply=5.2,5.3,5.4 -Xcommon=rh850 main.c
```

-Xmisra2012 [Professional Edition only]

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

[Specification format]

```
-Xmisra2012=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*.
- This option cannot be specified simultaneously with the -Xmisra2004 option.
- 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. [V2.02.00 or later]
  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.
- The __fp16 type is handled as the float type during the check. For the effects of this handling, see the description of the -Xcheck_language_extension [Professional Edition only] option.

[Example of use]

- To check the source code against MISRA-C:2012 rule number: 5.2, 5.3, and 5.4, describe as:

```
>ccrh -Xmisra2012=apply=5.2,5.3,5.4 -Xcommon=rh850 main.c
```

-Xignore_files_misra [Professional Edition only]

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

[Specification format]

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

- Interpretation when omitted
  All C source files are checked.

[Detailed description]

- This option does not check the source code of the file specified by *file* against the MISRA-C:2004 or MISRA-C:2012 rules.
- This option is valid only when the -Xmisra2004 or -Xmisra2012 option is specified.
  If the -Xmisra2004 or -Xmisra2012 option is not specified, this option will be ignored (A warning will not be output).

[Example of use]

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

```
>ccrh -Xmisra2004=all -Xignore_files_misra=sample.c -Xcommon=rh850 main.c
```

-Xcheck_language_extension [Professional Edition only]

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

[Specification format]

```
-Xcheck_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 checking of the MISRA-C:2004 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 (MISRA-C:2004 rule 8.1, MISRA-C:2012 rule 8.4) and #pragma interrupt is specified for it.
- This option is valid only when the -Xmisra2004 option or -Xmisra2012 option is specified.
  If the -Xmisra2004 option or -Xmisra2012 option is not specified, this option will be ignored (A warning will not be output).
  The __fp16 type is handled as the float type during the check. This affects the following rules.
  - MISRA-C:2004 rule 10.2
  - MISRA-C:2004 rule 10.3
  - MISRA-C:2004 rule 10.4

[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:

```
>ccrh -Xmisra2004=all -Xcheck_language_extension -Xcommon=rh850 main.c
```

-misra_intermodule [Professional Edition only] [V2.01.00 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

[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 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.
  5.1 5.6 5.7 5.8 5.9
  8.3 8.5 8.6
- .{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 this option is specified more than once, the last specification is valid. At this time, a warning will be output.
- This option will be ignored unless the -Xmisra2012 option is specified at the same time. At this time, a warning will be output.
- An error will occur in the following case.
  - When the parameter is omitted

[Remark]

- 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 from a project or its file name is changed, delete *file* and recheck source code against the MISRA-C:2012 rules.
- If there are many files to be checked and the symbol information to be stored in *file* is huge, the compilation speed gets slower.
- 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.

[Example of use]

- To check source code in a1.c, a2.c, and a3.c, describe as:

```
>ccrh -Xmisra2012=all -misra_intermodule=info.mi a1.c a2.c a3.c
```

-Xuse_fp16 [Professional Edition only] [V1.05.00 or later]

This option enables the half-precision floating-point type.

[Specification format]

```
-Xuse_fp16[=value]
```

- Interpretation when omitted
  The half-precision floating-point type is disabled.

[Detailed description]

- This option enables the half-precision floating-point type, which is a unique type extended from the C language standard.
- Specify "on" or "off" in *value*. If *value* is omitted, it is assumed that "on" has been specified.
- In the following cases, a warning will be output and this option will be ignored.
  - This option is specified simultaneously with the -strict_std.
  - This option is specified simultaneously with the -Xcpu=g3k option.
  - This option is specified simultaneously with the -Xfloat=soft option.
  - This option is specified simultaneously with the -Xround=zero option.

[Example of use]

- To enable the __fp16 type, describe as:

```
>ccrh -Xuse_fp16 main.c
```

Japanese/Chinese character control

The Japanese/Chinese character control option is as follows.

- -Xcharacter_set

-Xcharacter_set

This option specifies the Japanese/Chinese character code.

[Specification format]

```
-Xcharacter_set=code
```

- 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 source file.
- The items that can be specified as *code* 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 source 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 *code* 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:

```
>ccrh -Xcharacter_set=euc_jp -Xcommon=rh850 main.c
```

Optimization specification

The optimization specification options are as follows.

- -O
- -Xintermodule
- -Xinline_strcpy
- -Xmerge_string
- -Xalias
- -Xmerge_files
- -Xwhole_program
- -library [V2.00.00 or later]
- -goptimize [V2.01.00 or later]

-O

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

[Specification format]

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

- Interpretation when omitted
  Optimization that debugging is not affected is performed (It is the same result as when -Odefault option is specified).

[Detailed description]

- This option specifies the optimization level or the details of each optimization items.
- The items that can be specified as *level* are shown below.
  An error will occur if any other item is specified.

nothing	Optimization with debugging precedence Regards debugging as important and suppresses all optimization including default optimization.
default	Default Performs optimization that debugging is not affected (optimization of expressions and register allocation, and the like).
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 shortening the execution speed as important and performs the maximum optimization that is effective for general programs.

- If *level* and *item* are omitted, it is assumed that "size" has been specified.
- 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 4 has been specified. If the -Ospeed option is specified, this item is assumed that the -Ounroll=4 option is specified.

Optimization Item (item)	Parameter (value)	Description
inline	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, it is assumed that 2 has been specified. This item is valid when the -Osize or -Ospeed option is specified (when the -Osize option is specified, it is assumed that the -Oinline=3 option has been specified. When the -Ospeed option is specified, it is assumed that the -Oinline=2 option has been specified). If any of the -Osize, -Ospeed or -Oinline option is not specified, this item is assumed that the -Oinline=1 option is specified. If the -Onothing option is specified, this item is assumed that the -Oinline=0 option is specified.
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. If *value* is omitted, it is assumed that 100 has been specified. This item is valid when the -Oinline=2 option is specified (including when the -Ospped option is specified).
inline_init [V1.07.00 or later]	on or off	Using immediate value as initializer of auto variables If "on" is specified, auto variables are always initialized by assigning an immediate value. If "off" is specified, the compiler automatically selects to perform initialization by copying a value between memories or assigning an immediate value. If *value* is omitted, it is assumed that "on" has been specified.
delete_static_func	on or off	Deleting unused static functions If *value* is omitted, it is assumed that "on" has been specified. This item is valid when the -Onothing option is not specified.
pipeline	on or off	Pipeline optimization If *value* is omitted, it is assumed that "on" has been specified.
tail_call	on or off	End call optimization When "on" is specified, if there is a function call at the end of a function and certain conditions are met, a function call for that call is converted to an unconditional branch. The lp store/restore 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. This item is valid when the -Ospeed or -Osize option is specified.

Optimization Item (item)	Parameter (value)	Description
map	file name	External variable access optimization Base addresses are set in accordance with the external symbol allocation information generated by the linker, and code for accessing external and static variables relative to the base addresses is generated. When symbol "__gp_data" is defined and a value is specified in gp in the code of the startup routine, code for accessing variables relative to gp when possible is generated in accordance with the external symbol allocation information. Specify the external symbol allocation information file generated by the optimizing linker as *file name*. If *file name* is omitted, it performs linking once, and then after creating the external symbol allocation information file, repeats the process from compilation to linking.
smap	None	Optimization of access to external variables defined in the compilation unit Base address is set for external and static variables defined in the file to be compiled, and code that accesses these relative to the base address is generated.
align [V2.03.00 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 item is valid when the -Osize or -Ospeed option is specified. If the -stuff option is specified at the same time, this item is invalid and the operation is the same as that when "off" is specified.

- If this option is specified more than once for the same *item*, the option specified last will be valid.
- If *-Olevel* is specified following *-Oitem*, *-Oitem* which was specified first will be invalid. Note, however, that *-Omap* or *-Osmap* will not be affected by *-Olevel*.
- When -O*item* is not specified, the optimization items are interpreted as follows according to the -O*level* setting.

Optimization Item (*item*)	Optimization Level (*level*)			
	nothing	default	size	speed
unroll	1	1	1	4
inline	0	1	3	2
inline_size	-	-	-	100
inline_init	off	off	off	on
delete_static_func	off	on	on	on
pipeline	off	off	off	on
tail_call	off	off	on	on
map	-	-	-	-
smap	-	-	-	-
align	off	off	on[Note]	on

Note If the -misalign option is specified at the same time, "off" is assumed.

This table does not ensure that the optimization result is the same between the case where an optimization level is selected and then each optimization item setting is changed to the value shown for another level in this table and the case where the latter level is specified from the beginning. For example, the code output by specifying "-Ospeed" may not be the same as that output by specifying "-Osize -Ounroll=4 -Oinline=2 -Oinline_size=100 -Opipeline=on".

[Example of use]

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

```
>ccrh -Osize -Xcommon=rh850 main.c
```

-Xintermodule

This option performs inter-module optimization.

[Specification format]

```
-Xintermodule
```

- Interpretation when omitted
  Inter-module optimization is not performed.

[Detailed description]

- This option performs inter-module optimization.
- The main optimization contents are shown below.
  - Optimization using inter-procedural alias analysis
    The example of the output code is shown below.

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

[Output assembler source]
_func1.1:
    .stack  _func1.1 = 0
    mov     #_x, r2
    st.w    r0, 0x00000000[r2]
    mov     0x00000001, r5
    st.w    r5, 0x00000004[r2]
    mov     0x00000000, r10     ; 0 is directly assigned because a and b point to
different addresses.
    jmp     [r31]
_func2:
    .stack  _func2 = 0
    mov     #_x, r6
    addi    0x00000004, r6, r7
    br9     _func1.1
```

  - Constant propagation of parameters and return values
    The example of the output code is shown below.

```
[C source]
static int func(int x, int y, int z) {
    return x-y+z;
}
int func2() {
    return func(3,4,5);
}

[Output assembler source]
_func.1:
    .stack  _func.1 = 0
    mov     0x00000000, r10     ; "4(=3-4+5)" is assigned directly.
    jmp     [r31]
_func2:
    .stack  _func2 = 0
    mov     0x00000005, r8
    mov     0x00000004, r7
    mov     0x00000003, r6
    br9     _func.1
```

[Example of use]

- To perform inter-module optimization on source files "main.c" and "sub.c", describe as:

```
>ccrh -Xintermodule -Osize -Xcommon=rh850 main.c sub.c
```

-Xinline_strcpy

This option performs inline expansion of standard library functions "strcpy", "strcmp", "memcpy", and "memset" calls.

[Specification format]

```
-Xinline_strcpy
```

- Interpretation when omitted
  Inline expansion of standard library functions "strcpy", "strcmp", "memcpy", and "memset" calls is not performed.

[Detailed description]

- This option performs inline expansion of standard library functions "strcpy", "strcmp", "memcpy", and "memset" calls.
- This option can not be specified together with the -Xpack option.
- Inline expansion of strcpy is performed only when the second argument is a character string.
- If this option is specified, arrays and character strings are allocated automatically to 4-byte boundary area.
- This improves the execution speed of the program to be generated, but it increases the code size.

[Example of use]

- To perform inline expansion of standard library functions "strcpy", "strcmp", "memcpy", and "memset" calls, describe as:

```
>ccrh -Xinline_strcpy -Xcommon=rh850 main.c
```

-Xmerge_string

This option merges string literals.

[Specification format]

```
-Xmerge_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. However, if a different section is specified, the section to which the string literal is allocated will depend on the order of appearance in the source.

[Example of use]

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

```
>ccrh -Xmerge_string -Xcommon=rh850 main.c
```

-Xalias

This option performs optimization with consideration for the type of the data indicated by the pointer.

[Specification format]

```
-Xalias=value
```

- Interpretation when omitted
  Optimization with consideration for the type of the data indicated by the pointer, based on the ANSI standard is not performed.

[Detailed description]

- This option specifies whether to perform optimization with consideration for the type of the data indicated by the pointer, based on the ANSI standard.
- The items that can be specified as *value* are shown below.
  An error will occur if any other item is specified.

ansi	Optimization with consideration for the type of the data indicated by the pointer, based on the ANSI standard is performed.
noansi	Optimization with consideration for the type of the data indicated by the pointer, based on the ANSI standard is not performed.

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

[Example of use]

- To perform optimization with consideration for the type of the data indicated by the pointer, based on the ANSI standard, describe as:

```
>ccrh -Xalias=ansi -Xcommon=rh850 -Osize main.c
```

-Xmerge_files

This option merges two or more C source files and compiles them.

[Specification format]

```
-Xmerge_files
```

- Interpretation when omitted
  Compilation is performed at the input-file level, without merging.

[Detailed description]

- This option merges two or more C source files and compiles them. And then it outputs one file.
- If the -o option is specified, then the specified file name is used for the output file. If the -o option is not specified, then the file name is in accordance with the interpretation of the -o option being omitted for the initially specified C source file.
- If one C source file is input and if this option is specified together with the -P option, this option will be invalid.
- If this option is specified at the same time as the -S or -c option, then for the 2nd and subsequent C source files that are specified, an empty file is output in accordance with the interpretation that the -o option was omitted.
- If this option is specified at the same time with the -Oinline option, inline expansion is performed between files.
- Operation is not guaranteed if an object file is generated with this option specified and any of link options -delete, -rename, and -replace is specified at linkage of the object file.

[Example of use]

- To merge main.c and sub.c, compile them, and output one file, describe as:

```
>ccrh -Xmerge_files -Xwhole_program -Xcommon=rh850 -Osize main.c sub.c
```

-Xwhole_program

This option performs optimization assuming that the files to be compiled comprise the entire program.

[Specification format]

```
-Xwhole_program
```

- Interpretation when omitted
  It is not assumed that the files to be compiled comprise the entire program.

[Detailed description]

- This option performs optimization assuming that the files to be compiled comprise the entire program.
- 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.
  - Even 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 -Xintermodule option is specified.
  If two or more C source files are input, it is assumed that the -Xmerge_files option is specified.

[Example of use]

- To perform optimization assuming that the files to be compiled comprise the entire program, describe as:

```
>ccrh -Xwhole_program -Xcommon=rh850 main.c
```

-library [V2.00.00 or later]

This option performs inline expansion for calling the standard library functions.

[Specification format]

```
-library={function|intrinsic}
```

- Interpretation when omitted
  It has the same meaning as when *function* is specified.

[Detailed description]

- This option controls whether to perform function call or inline expansion for the following standard library functions.
  - abs(), labs(), llabs()
  - fabs(), fabsf()
  - sqrt(), sqrtf()
  - fmax(), fmaxf()
  - fmin(), fminf()
  - copysign(), copysignf()
- If *function* is specified, a code to always call the target functions is generated.
- If *intrinsic* is specified, inline expansion is performed for calling the target functions if possible.
- The parameter must be specified in lowercase characters.
- If this option is specified more than once, the last specification is valid.
- An error will occur in any of the following cases.
  - When the parameter is omitted
  - When a parameter other than *function* or *intrinsic* is specified as the parameter
- If inline expansion is performed for calling the target library functions because of this option, the expanded code will not update variable errno. The operation for the following inputs differs from that when a function was called.
  - sqrt or sqrtf: -0.0, negative number, or not-a-number
  - fmax, fmaxf, fmin, or fminf: +0.0, -0.0, or not-a-number

-goptimize [V2.01.00 or later]

This option generates information for link-time optimization.

[Specification format]

```
-goptimize
```

- Interpretation when omitted
  None

[Detailed description]

- This option generates additional information used at link-time optimization in the output file.
- When this option is specified for a file, link-time optimization will be applied at linkage of the file.
  For details on link-time optimization, see the description of the link option -OPtimize.

Generated code control

The generated code control options are as follows.

- -Xpack
- -misalign [V2.04.00 or later]
- -Xbit_order
- -Xpass_source
- -Xswitch
- -Xreg_mode
- -Xreserve_r2
- -r4 [V1.07.00 or later]
- -Xep
- -Xfloat
- -Xfxu [V2.00.00 or later]
- -Xcall_jump
- -Xfar_jump
- -Xdiv
- -Xcheck_div_ov
- -relaxed_math [V2.00.00 or later]
- -Xuse_fmaf
- -use_recipf [V2.00.00 or later]
- -approximate [V2.02.00 or later]
- -Xunordered_cmpf
- -Xmulti_level
- -Xpatch
- -Xdbl_size
- -Xround
- -Xalign4
- -Xstack_protector/-Xstack_protector_all [Professional Edition only]
- -Xsection
- -stuff [V2.03.00 or later]
- -Xcheck_exclusion_control [V1.04.00 or later]
- -Xresbank_mode [V2.00.00 or later]
- -insert_dbtag_with_label [V1.06.00 or later]
- -store_reg [Professional Edition only] [V1.06.00 or later]
- -control_flow_integrity [Professional Edition only] [V1.07.00 or later]
- -pic [V1.07.00 or later]
- -pirod [V1.07.00 or later]
- -pid [V1.07.00 or later]

-Xpack

This option performs the structure packing.

[Specification format]

```
-Xpack=num
```

- Interpretation when omitted
  The structure packing is not performed.

[Detailed description]

- This option performs the structure packing.
- If this option is specified, struct members will not be aligned by their member types, but rather code will be generated with alignment packed to the specified *num* bytes.
- 1, 2, or 4 can be specified as *num*.
  An error will occur if any other item is specified.
- This option can not be specified together with the -Xinline_strcpy option.
- If this option is specified when the structure packing is specified by the #pragma directive in the C source, the value specified by this option is applied to all structures until the first #pragma directive appears.
  After that, the value of the #pragma directive is applied.
  Even after the #pragma directive has appeared, however, the value specified by the option is applied if the default value is specified (if the value of the packing by the #pragma directive).

[Example of use]

- To generate code with struct member alignment packed to 1 byte, describe as:

```
>ccrh -Xpack=1 -Xcommon=rh850 main.c
```

-misalign [V2.04.00 or later]

This option generates an instruction string that performs a misaligned memory access.

[Specification format]

```
-misalign
```

[Detailed description]

- For memory accesses, this option generates a more effective instruction string assuming that the device supports access to unaligned addresses.
- Specifying this option more than once has the same effect as specifying it once only. No warning is output in this case.

[Remark]

- To specify this option, enable the misaligned memory access function of the device. For details, see the user's manual of the device.

-Xbit_order

This option specifies the order of bit-field members.

[Specification format]

```
-Xbit_order=pos
```

- Interpretation when omitted
  The bit-field members are allocated from the lower bit.

[Detailed description]

- This option specifies the order of bit-field members.
- The items that can be specified as *pos* are shown below.
  An error will occur if any other item is specified.

left	The members are allocated from the upper bit.
right	The members are allocated from the lower bit.

- An error will occur if *pos* is omitted.
- If this option is specified when the order of bit-field members is specified by the #pragma directive in the C source, the value specified by this option is applied to all members until the first #pragma bit_order directive appears.
  After that, the value of the #pragma directive is applied.

[Example of use]

- To allocate the bit-field members from the upper bit, describe as:

```
>ccrh -Xbit_order=left -Xcommon=rh850 main.c
```

-Xpass_source

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

[Specification format]

```
-Xpass_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:

```
>ccrh -Xpass_source -S -Xcommon=rh850 main.c
```

-Xswitch

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

[Specification format]

```
-Xswitch=type
```

- Interpretation when omitted
  ccrh 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 items that can be specified as *type* are shown below.
  An error will occur if any other item 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.
table	Outputs the code in a table jump format. References a table indexed on the values in the case statements, and selects and processes case labels from the switch statement values. The code will branch to all the case statements with about the same speed. However, if case values are not used in succession, an unnecessary area will be created. If the difference between the maximum and minimum values of the case labels exceeds 8192, then this option is ignored, and the optimum output format for each switch statement is selected automatically.

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

[Example of use]

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

```
>ccrh -Xswitch=binary -Xcommon=rh850 main.c
```

-Xreg_mode

This option specifies the register mode.

[Specification format]

```
-Xreg_mode=mode
```

- Interpretation when omitted
  The 32-register mode object file is generated.

[Detailed description]

- This option generates the object file for the specified register mode.
- This option limits the number of registers used by ccrh to 32 (the 32-register mode) or 22 (the 22-register mode or register mode "common") and embeds the magic number into the object file.
- Use register mode "common" to generate the object file that does not depend on register modes.
- The items that can be specified as *mode* are shown below.
  An error will occur if any other item is specified.

Register Mode (*mode*)	Working Registers	Registers for Register Variables
common	r10 to r14	r25 to r29
22	r10 to r14	r25 to r29
32	r10 to r19	r20 to r29

- An error will occur if *mode* is omitted.
- This option generates the code using the register that can be used for a C source file.
- If 32-register mode object files and 22-register mode object files are mixed, an error will occur at linkage.

[Example of use]

- To generate the 22-register mode object file, describe as:

```
>ccrh -Xreg_mode=22 -Xcommon=rh850 main.c
```

-Xreserve_r2

This option reserves the r2 register.

[Specification format]

```
-Xreserve_r2
```

- Interpretation when omitted
  The compiler uses the r2 register without reserving it.

[Detailed description]

- This option reserves the r2 register and generates code that does not use this register by the compiler.

[Example of use]

- To reserve the r2 register and generates code that does not use this register by the compiler, describe as:

```
>ccrh -Xreserve_r2 -Xcommon=rh850 main.c
```

-r4 [V1.07.00 or later]

This option specifies how to handle the r4 register.

[Specification format]

```
-r4=mode
```

- Interpretation when omitted
  The value of the r4 register is fixed for the entire project.

[Detailed description]

- This option specifies how to handle the r4 register.
- The items that can be specified as *mode* are shown below.
  An error will occur if any other item is specified.

fix	Fixes the value of the r4 register for the entire project. Specify this parameter when GP-relative sections[Note] are used.
none	The compiler does not use the r4 register.

Note See "4.2.6.1 Allocation of function and data to section" for details about GP-relative sections.

-Xep

This option specifies how to handle the ep register.

[Specification format]

```
-Xep=mode
```

- Interpretation when omitted
  The ep register is treated as a register guaranteeing the value before and after the function call.

[Detailed description]

- This option specifies how to handle the ep register.
- The items that can be specified as *mode* are shown below.
  An error will occur if any other item is specified.

fix	Fixes the value of the ep register for the entire project. Specify this parameter when EP-relative sections[Note] are used.
callee	Treats the ep register as a register guaranteeing the value before and after the function call. Specify this parameter when the -Omap or -Osmap option is specified.

Note See "4.2.6.1 Allocation of function and data to section" for details about GP-relative sections.

- An error will occur if *mode* is omitted.
- The same specification must be made for all source files.
  A different specification cannot be made for each source file.
  If there are object files with different specifications, an error will occur at linkage.

[Example of use]

- To fix the value of the ep register for the entire project, describe as:

```
>ccrh -Xep=fix -Xcommon=rh850 main.c
```

-Xfloat

This option controls generating floating-point calculation instructions.

[Specification format]

```
-Xfloat=type
```

- Interpretation when omitted
  If the-Xcpu=g3k option is specified, -Xfloat=soft is assumed.
  In any other case, -Xfloat=fpu is assumed.

[Detailed description]

- This option controls generating floating-point calculation instructions.
- The items that can be specified as *type* are shown below.
  An error will occur if any other item is specified.

soft	Generates runtime function call instructions for floating-point calculations.
fpu	Generates floating-point calculation instructions of FPU (floating-point unit) for floating-point calculations. However, if -Xcpu=g3kh is specified, runtime function call instructions for double-precision calculations will be generated. If this option is specified together with the -Xcpu=g3k option, this option is invalid and -Xfloat=soft is assumed.

- An error will occur if *type* is omitted.
- If this option is specified together with the -Xcpu=g3k option, this option will be invalid and runtime function call instructions will always be generated.
- If soft is specified as *type*, the -Xround=zero option will be invalid and -Xround=nearest will always be valid.

[Example of use]

- To generate the jarl32 and jr32 instructions for function-call branches, describe as:

```
>ccrh -Xfloat=soft -Xcommon=rh850 main.c
```

-Xfxu [V2.00.00 or later]

This option controls usage of FXU instructions.

[Specification format]

```
-Xfxu[={on|off}]
```

- Interpretation when omitted
  If -Xcpu=g4mh is specified, it is regarded that -Xfxu=on is specified.
  If an option other than -Xcpu=g4mh is specified, it is regarded that -Xfxu=off is specified.

[Detailed description]

- This option controls how to handle the system registers for FXU in exception handlers.
- If *on* is specified, FXU instructions are regarded to be used in the program.
  If *off* is specified, FXU instructions are regarded not to be used in the program.
- If the parameter is omitted, it has the same meaning as when *on* is specified.
- If this option is specified more than once, the last specification is valid.
- If the specification of this option is changed for each source file, registers may not be correctly managed in exception handlers. The specification must be the same for all source files.
- If an option other than -Xcpu=g4mh is specified, this option is ignored. A warning is output in this case.
- An error will occur in the following case.
  - When a parameter other than *on* or *off* is specified

[Remark]

In V2.00.00, FXU instructions are not generated even if this option is specified. Only codes generated by exception handlers will be affected by this option.

-Xcall_jump

This option controls generating function-call branch instructions.

[Specification format]

```
-Xcall_jump=num
```

- Interpretation when omitted
  The jarl and jr instructions are generated for function-call branches.

[Detailed description]

- This option controls generating function-call branch instructions.
- The items that can be specified as *num* are shown below.
  An error will occur if any other item is specified.

22	Generates the jarl and jr instructions for the branch to the function.
32	Generates the jarl32 and jr32 instructions for the branch to the function.

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

[Example of use]

- To generate the jarl32 and jr32 instructions for the branch to the function, describe as:

```
>ccrh -Xcall_jump=32 -Xcommon=rh850 main.c
```

-Xfar_jump

This option controls outputting far jump.

[Specification format]

```
-Xfar_jump=file
```

- Interpretation when omitted
  The instructions in accordance with the -Xcall_jump option are generated.

[Detailed description]

- This option generates the code that uses instructions with a branch distance of 32 bits for the branch to functions specified in far jump calling function list file *file*.
- ".fjp" is recommended as the extension of *file*.
- An error will occur if *file* does not exist.
- An error will occur if *file* is omitted.
- An error will occur at linkage if the distance between a branch instruction and a branch destination function exceeds 22 bits (±2 Mbytes) when the -Xcall_jump=22 option is specified. In this case, recompile by using this option.
- If this option is specified more than once, the option specified last will be valid.
- The example of the output code is shown below.
  - C source

```
far_func();     /* "jarl _far_func, lp" is output by default. */
```

  - Output assembly source

```
jarl32  _far_func, lp
```

Remark Cautions about are the format of the far jump calling function list file as follows.

- Describe with one function name per line.
  If two or more function name is described, the first name will be valid.
- Describe the function name (label name in an assembly source) by prefixing "_" to that in C language.
  However, the following formats can be specified instead of function names.

Format	Meaning
{all_function}	All functions are called.

- Not only functions that are called from a C source file, but operation runtime functions can also be specified.
  When specifying an operation runtime function, instead of prefixing "_" to the function name, specify the function name as it is in Table 7.16 in "7.4.13 Operation runtime functions".
- A space and tab can be inserted before and after function names.
- Only ASCII characters can be used.
  After the space characters at the beginning of a line, a non-space character string until the next space character or the end of the line is treated as a function name, and the rest of the line (from the space character to the end of the line) is ignored.

- Comments cannot be inserted.
- Up to 1023 characters can be specified per line (including a space and tab).

The example when specifying functions is shown below.

```
_func_led
_func_beep
_func_motor
    :
_func_switch
_COM_div64
```

[Example of use]

- To generate the code that uses an instruction with a branch distance of 32 bits for the branch to the function specified in func.fjp, code as:

```
>ccrh -Xfar_jump=func.fjp -Xcommon=rh850 main.c
```

-Xdiv

This option generates the div and divu instructions for division.

[Specification format]

```
-Xdiv
```

- Interpretation when omitted
  The divq and divqu instructions are generated for division.

[Detailed description]

- This option generates the div and divu instructions instead of the divq and divqu instructions for division.
- Although the divq and divqu instructions are fast, the number of execution cycles will differ depending on the values of the operands.
  For this reason, specify this option if it is necessary to maintain a constant number of execution cycles at all times (e.g. in order to guarantee real-time performance).

[Example of use]

- To generate the div and divu instructions for division, describe as:

```
>ccrh -Xdiv -Xcommon=rh850 main.c
```

-Xcheck_div_ov

This option checks the OV flag at division.

[Specification format]

```
-Xcheck_div_ov=num
```

- Interpretation when omitted
  Code that does not check the OV flag at division is generated.

[Detailed description]

- This option generates code (fetrap instruction) that checks the OV flag after division instructions and generate an FE level software exception when the OV flag is 1.
- The value that can be specified for *num* is 1 to 15 (a value that can be specified for the operand of the fetrap instruction).
  An error will occur if any other item is specified.
- An error will occur if *num* is omitted.

[Example of use]

- To check the OV flag at division, describe as:

```
>ccrh -Xcheck_div_ov=1 -Xcommon=rh850 main.c
```

-relaxed_math [V2.00.00 or later]

This option generates a floating-point calculation code with efficiency given precedence over strictness.

[Specification format]

```
-relaxed_math
```

- Interpretation when omitted
  None

[Detailed description]

- For floating-point calculations, this option generates a calculation code that is not in strict accordance with the C-language standard or IEEE754, but is efficient with respect to the code size and execution speed.
- The following options are assumed to be simultaneously specified.
  - -Xuse_fmaf
  - -use_recipf
  - -approximate [V2.02.00 or later]
- If this option is specified more than once, it has the same meaning as when this option is specified once. No warning is output in this case.

[Remark]

When this option is specified, floating-point calculations are performed in the following manner and sometimes the operation result will differ from that obtained by calculations strictly in accordance with the C-language standard or IEEE754.

- The meaning of the sign of 0.0 is ignored.
- Expressions are deformed by using algebraic characteristics, assuming that an exception or precision error is not generated by calculations.
- It is assumed that a calculation does not result in I/O of a NaN or infinity in a comparison calculation or other calculations.
  A program that handles these values might cause an unexpected execution result. Therefore, care must be taken when using this option.

  Example:
  Generally, if x or y is a NaN, this program does not call function func2.
  However, if this option is specified, an efficient code[Note] is generated instead of assuming NaN input. As a result, function func2 may be called.

Note The performance remarkably changes when the -Xfloat=soft option is specified.

```
void func1(double x, double y) {
  if (x < y) {
    func2();
  }
}
```

-Xuse_fmaf

This option generates product-sum operation instructions.

[Specification format]

```
-Xuse_fmaf
```

- Interpretation when omitted
  Product-sum operation instructions are not generated.

[Detailed description]

- This option generates product-sum operation instructions (fmaf.s, fmsf.s, fnmaf.s, and fnmsf.s) for single-precision floating-point product-sum operations.
- Specifying this option will accelerate the execution speed but change the operation precision.

[Example of use]

- To generate product-sum operation instructions for single-precision floating-point product-sum operations, describe as:

```
>ccrh -Xuse_fmaf -Xcommon=rh850 main.c
```

-use_recipf [V2.00.00 or later]

This option generates the recipf instructions.

[Specification format]

```
-use_recipf
```

- Interpretation when omitted
  None

[Detailed description]

- This option generates the recipf.d and recipf.s instructions.
- If this option is specified more than once, it has the same meaning as when this option is specified once. No warning is output in this case.
- If usage of the FPU is not enabled by the -Xcpu option or -Xfloat option, this option is ignored. No warning is output in this case.

[Remark]

If the recipf instructions are generated by this option, the operation result may differ from that when this option is not specified.

Since a recipf instruction always triggers an incorrect operation exception of the FPU, FPU exception processing should be set appropriately.

-approximate [V2.02.00 or later]

This option replaces floating-point calculations with equivalent approximate calculations.

[Specification format]

```
-approximate
```

- Interpretation when omitted
  None

[Detailed description]

- This option replaces floating-point calculations with equivalent approximate calculations.
  This replacement generates a calculation code that is efficient with respect to the code size and execution speed.

[Remark]

When this option is specified, floating-point calculations are performed in the following manner and sometimes the operation result will differ from that obtained by calculations strictly in accordance with the C-language standard or IEEE754.

- The meaning of the sign of 0.0 is ignored. Expressions are deformed by using algebraic characteristics, assuming that an exception or precision error is not generated by calculations.

-Xunordered_cmpf

This option detects invalid operation exceptions in floating-point comparison.

[Specification format]

```
-Xunordered_cmpf
```

- Interpretation when omitted
  In floating-point comparison, invalid operation exceptions are not detected when qNaN is included.

[Detailed description]

- This option generates code by using the comparison condition for generating an invalid operation exception when any of the comparison values is a qNaN in floating-point comparison.
- This option is valid only for floating-point comparison using instructions of FPU (floating-point unit).

[Example of use]

- To detect invalid operation exceptions in floating-point comparison, describe as:

```
>ccrh -Xunordered_cmpf -Xcommon=rh850 main.c
```

-Xmulti_level

This option specifies the generation of a multi-core program.

[Specification format]

```
-Xmulti_level=level
```

- Interpretation when omitted
  A single-core program is generated.

[Detailed description]

- This option generates a program for the specified core.
- The items that can be specified as *level* are shown below.
  An error will occur if any other item is specified.

0	Generates a single-core program. The #pragma pmodule directives in the program are ignored.
1	Generates a multi-core program. The #pragma pmodule directives in the program become valid.

[Example of use]

- To generate a multi-core program, describe as:

```
>ccrh -Xmulti_level=1 -Xcommon=rh850 file1.c file2.c
```

-Xpatch

This option applies a patch.

[Specification format]

```
-Xpatch=string[,string]...
```

- Interpretation when omitted
  The default patch is applied.

[Detailed description]

One of the following can be specified as *string*. An error will occur if any other item is specified.

- dw_access
  A code is generated without using the ld.dw and st.dw instructions.
- switch [V1.03.00 or later]
  If -Xcpu=g3m is specified without specifying this option, generation of the switch instruction is suppressed. Specifying this option cancels the suppression and generates the switch instruction.
- syncp [V1.03.00 or later]
  If -Xcpu=g3m is specified simultaneously, syncp instruction is inserted at the entry of each interrupt function defined with the #pragma interrupt directive in which priority=SYSERR/FPI/FENMI/FEINT/EIINT_PRIORITYX (X: 0 to 15) is specified or neither priority nor channel is specified.
  If an option other than -Xcpu=g3m is specified, this option is ignored.
- br [V2.04.00 or later]
  If -Xcpu=g3m is specified without specifying this option, generation of br disp9 instructions that satisfy specific conditions is suppressed. Specifying this option cancels the suppression and generates the br disp9 instruction.
  If an option other than -Xcpu=g3m is specified, generation of the br disp9 instruction is not suppressed regardless of the specification of this option.
- br_jr [V2.04.01 or later]
  If -Xcpu=g3kh is specified at the same time, generation of br disp9, jr disp22, and jr disp32 instructions is suppressed.
  If an option other than -Xcpu=g3kh is specified, this option is ignored.

The following shows the default patch that is applied when this option is not specified.

- When -Xcpu=g3m is specified, the following patch is applied:
  - Suppressing generation of the switch instruction
  - Suppressing generation of br disp9 instructions that satisfy specific conditions

[Example of use]

- To generate a code without using the ld.dw and st.dw instructions, describe as:

```
>ccrh -Xpatch=dw_access -Xcommon=rh850 main.c
```

-Xdbl_size

This option specifies the data size of double and long double type.

[Specification format]

```
-Xdbl_size=num
```

- Interpretation when omitted
  double and long double type are 8 bytes (this is the same result as when -Xdbl_size=8 is specified).

[Detailed description]

- This option specifies the data size of double and long double type.
- One of the following can be specified as *num*. An error will occur if any other item is specified.
  - 4
    double and long double type are 4 bytes.
  - 8
    double and long double type are 8 bytes.

-Xround

This option specifies the mode for rounding floating-point constants.

[Specification format]

```
-Xround=mode
```

- Interpretation when omitted
  Floating-point constants are rounded to the nearest representable values (this is the same result as when -Xround=nearest is specified).

[Detailed description]

- This option specifies the mode for rounding floating-point constants.
- One of the following can be specified as *mode*. An error will occur if any other item is specified.
  - nearest
    Floating-point constants are rounded to the nearest representable values.
  - zero
    Floating-point constants are rounded toward zero.
- If this option is specified together with the -Xfloat=soft option, this option will be invalid and floating-point constants will always be rounded to the nearest representable values.

-Xalign4

This option specifies the alignment value for branch destination addresses.

[Specification format]

```
-Xalign4[=mode]
```

- Interpretation when omitted
  The alignment value for branch destination addresses is set to 2.

[Detailed description]

- This option sets the alignment value for branch destination addresses specified by *mode* to 4.
- One of the following can be specified as *mode*. An error will occur if any other item is specified.
  - function
    The alignment value for function start addresses is set to 4.
  - loop
    The alignment value for function start addresses and the start addresses of all loops is set to 4.
  - innermostloop
    The alignment value for function start addresses and the start address of the innermost loop is set to 4.
  - all
    The alignment value for function start addresses and all branch destination addresses is set to 4.
  - When =*mode* is omitted
    The alignment value for function start addresses is set to 4 (same as function).
- If an object module file or a standard library that has been generated through compilation without using this option is specified for linkage, the warning W0561322 will be output at linkage but program execution will have no problem.

-Xstack_protector/-Xstack_protector_all [Professional Edition only]

This option specifies generation of a code for detection of stack smashing.

[Specification format]

```
-Xstack_protector[=num]
-Xstack_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 code for detection of stack smashing indicates the instructions for executing the three processes shown below.
  (1) A 4-byte area is allocated just before the local variable area (in the direction towards address 0xFFFFFFFF) at the entry to a function, and the value specified by *num* is stored in the allocated area.
  (2) At the end of the function, whether the 4-byte area in which *num* was stored has been rewritten is checked.
  (3) If the value has been rewritten in (2), the __stack_chk_fail function is called as the stack has been smashed.
- A decimal number from 0 to 4294967295 should be specified in *num*. If the specification of *num* is omitted, the compiler automatically specifies the number.
- The __stack_chk_fail function needs to be defined by the user and the processing to be executed upon detection of stack smashing should be written. Note the following items when defining the __stack_chk_fail function.
  - The only possible type of return value is void and the __stack_chk_fail function does not have formal parameters.
  - Do not define the function as static.
  - It is prohibited to call the __stack_chk_fail function as a normal function.
  - The __stack_chk_fail function is not subject to generating a code for detection of stack smashing due to the -Xstack_protector and -Xstack_protector_all options and #pragma stack_protector.
  - Prevent returning to the caller, that is, the function where stack smashing was detected by taking measures such as calling abort() in the __stack_chk_fail function to terminate the program.
  - When calling another function in the __stack_chk_fail function, note that stack smashing is not detected recursively in the function that was called.
  - When this facility is used for a function for which PIC (see "8.6 PIC/PID Facility") is performed, PIC should also be performed for the __stack_chk_fail function.
- If -Xstack_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 -Xstack_protector_all is specified, this option generates a code for detection of stack smashing for all functions.
- If these options are used simultaneously with #pragma stack_protector, the specification by #pragma stack_protector becomes valid.
- Even though this option is specified, a code for detection of stack smashing is not generated for the functions for which one of the following #pragma directives is specified.
  #pragma inline, inline keyword, #pragma inline_asm, #pragma no_stack_protector

[Example of use]

- C source

```
#include <stdio.h>
#include <stdlib.h>

void f1()   // Sample program in which the stack is smashed
{
    volatile char str[10];
    int i;
    for (i = 0; i <= 10; i++){
      str[i] = i; // Stack is smashed when i=10
    }
}

void __stack_chk_fail(void)
{
    printf("stack is broken!");
    abort();
}
```

- Output codes
  When compilation is performed with -Xstack_protector=1234 specified.

```
_f1:
      .stack _f1 = 16
      add 0xFFFFFFF0, r3
       movea 0x000004D2, r0, r1  ; The specified <number> 1234 is stored in the stack
area.
      st.w r1, 0x0000000C[r3]
      mov 0x00000000, r2
      br9 .BB.LABEL.1_2
.BB.LABEL.1_1:  ; bb
      movea 0x00000002, r3, r5
      add r2, r5
      st.b r2, 0x00000000[r5]
      add 0x00000001, r2
.BB.LABEL.1_2:  ; bb7
      cmp 0x0000000B, r2
      blt9 .BB.LABEL.1_1
.BB.LABEL.1_3:  ; return
      ld.w 0x0000000C[r3], r1   ; Data is loaded from the location where <number>
      movea 0x000004D2, r0, r12 ; was stored at the entry to a function and
      cmp r12, r1               ; it is compared with the specified <number> 1234.
      bnz9 .BB.LABEL.1_5        ; If they do not match, a branch occurs.
.BB.LABEL.1_4:  ; return
      dispose 0x00000010, 0x00000000, [r31]
.BB.LABEL.1_5:  ; return
      br9 ___stack_chk_fail     ; __stack_chk_fail is called.

___stack_chk_fail:
      .stack ___stack_chk_fail = 4
      prepare 0x00000001, 0x00000000
      mov #.STR.1, r6
      jarl _printf, r31
      jarl _abort, r31
      dispose 0x00000000, 0x00000001, [r31]
```

-Xsection

This option specifies the default sections for data.

[Specification format]

```
-Xsection=string=value[,string=value]
```

- Interpretation when omitted
  The default section is set to .bss for uninitialized data, .data for initialized data, or .const for constant data.

[Detailed description]

- This option specifies the default section attributes for data.
- The following shows the character strings that can be specified for *string* and *value*, and the default sections for each setting. An error will occur if any other item is specified for *string* and *value*.

string	value	Default Section		
		Uninitialized Data	Initialized Data	Constant Data
data	r0_disp16	.zbss	.zdata	-
	r0_disp23	.zbss23	.zdata23	-
	ep_disp16	.ebss	.edata	-
	ep_disp23	.ebss23	.edata23	-
	gp_disp16	.sbss	.sdata	-
	gp_disp23	.sbss23	.sdata23	-
const	zconst	-	-	.zconst
	zconst23	-	-	.zconst23
	pcconst16 [V1.07.00 or later]	-	-	.pcconst16
	pcconst23 [V1.07.00 or later]	-	-	.pconst23

- When the attribute is changed by #pragma section, the attribute specified by #pragma section will be valid.

Combinations of this option and other options which will cause an error are shown below.

-Xsection=data=ep_disp16 -Xsection=data=ep_disp23	An error will occur when specified simultaneously with -Omap. An error will occur when specified simultaneously with -Osmap.
-Xsection=data=gp_disp16 -Xsection=data=gp_disp23	An error will occur when specified simultaneously with -r4=none.
-Xsection=data=r0_disp16 -Xsection=data=r0_disp23	An error will occur when specified simultaneously with -pid.
-Xsection=const=zconst -Xsection=const=zconst23	An error will occur when specified simultaneously with -pirod.
-Xsection=const=pcconst16 -Xsection=const=pcconst23	An error will occur when not specified simultaneously with -pirod.

-stuff [V2.03.00 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 4, "_4" is not added to a section name.
  Examples:
  When the number of alignment of variables is 4: .bss
  When the number of alignment of variables is 2: .bss_2
  When the number of alignment of variables is 1: .bss_1

[Example of use]

```
const char c=1;
const short s=2;
const long l=3;
```

Default	-stuff specification

`.section .const, const`
`_c:`
`.db 0x01`
`.align 2`
`_s:`
`.dhw 0x0002`
`.align 4`
`_l:`
`.dw 0x00000003`
	`.section .const_1, const, align=1`
	`_c:`
	`.db 0x01`
	
	`.section .const_2, const, align=2`
	`.align 2`
	`_s:`
	`.dhw 0x0002`
	
	`.section .const, const`
	`.align 4`
	`_l:`
	`.dw 0x00000003`

[Remark]

- Each section name reflects the following options or specification in #pragma section:
  -Xsection, -Xmulti_level

-Xcheck_exclusion_control [V1.04.00 or later]

This option is used to select checking of exclusive control.

[Specification format]

```
-Xcheck_exclusion_control=<filename>
```

- Interpretation when omitted
  Checking of exclusive control is disabled.

[Detailed description]

- This option loads the setting file and inserts the dbtag instruction at the specified location.
- This function assumes usage via CS+ and it should not be used directly by the user.

-Xresbank_mode [V2.00.00 or later]

This option specifies the operating mode of the resbank instruction.

[Specification format]

```
-Xresbank_mode=num
```

- Interpretation when omitted
  It has the same meaning as when -Xresbank_mode=0 is specified.

[Detailed description]

- This option generates a code assuming that the resbank instruction will operate with the value specified in *num* being set in RBCR0.MD (register for specifying the save mode of the register bank).
- 0 or 1 can be specified for *num*. Specify the same value as that set in RBCR0.MD.
- If this option is specified more than once, the last specification is valid.
- If the specification of this option is changed for each source file, registers may not be correctly managed in exception handlers. The specification must be the same for all source files.
- This option is valid only in an exception handler in which resbank was specified by the #pragma interrupt directive.
- An error will occur in any of the following cases.
  - When *num* is omitted
  - When a value other than 0 or 1 is specified
  - When a parameter other than g4mh is specified in the -Xcpu option

[Remark]

Even though this option is specified, a code for setting a value to RBCR0.MD is not generated. The value must be directly set with the user program.

-insert_dbtag_with_label [V1.06.00 or later]

This option controls insertion of the dbtag instruction.

[Specification format]

```
-insert_dbtag_with_label=file, line, label, tagid
```

- Interpretation when omitted
  The dbtag instruction is not inserted.

[Detailed description]

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

-store_reg [Professional Edition only] [V1.06.00 or later]

This option controls detection of writing to control registers or insertion of synchronization processing between registers.

[Specification format]

```
-store_reg[=mode]
```

- Interpretation when omitted
  If #pragma register_group is written, the operation is the same as that when -store_reg=list is specified.

[Detailed description]

- This option recognizes #pragma register_group as a valid #pragma directive and carries out the operation specified by *mode*.
  See "4.2.6.14 Detection of writing to control registers or insertion of synchronization processing [Professional Edition only] [V1.06.00 or later]" for details on #pragma register_group.
- Specify either one of the following as *mode*. An error will occur if any other item is specified.
  - list
    This option allows the compiler to detect writing to the control registers defined as #pragma register_group and display the source line number of the write instructions to the standard error output, except where the succeeding instruction will clearly be for writing to the same group, in which case the compiler does not display the source line number.
  - list_all
    This option allows the compiler to detect writing to the control registers defined as #pragma register_group and display the source line number of the write instructions to the standard error output. The source line number are displayed regardless of whether the succeeding instruction will clearly be for writing to the same group.
  - sync
    This option allows the compiler to detect writing to the control registers defined as #pragma register_group and inserts synchronization processing after write instructions for these registers, except where the succeeding instruction will clearly be for writing to the same group, in which case the compiler does not insert a synchronization processing.
  - ignore
    #pragma register_group is ignored but a warning is not output.
  - When =*mode* is omitted
    The operation is the same as that when -store_reg=list is specified.

-control_flow_integrity [Professional Edition only] [V1.07.00 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.
  void __control_flow_integrity(void *addr);
  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.
  - 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.
  - If the -pic option is specified at the same time, an error will occur.

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 -S -control_flow_integrity is specified for compilation

```
___control_flow_chk_fail:
    .stack ___control_flow_chk_fail = 4
    prepare 0x00000001, 0x00000000
    jarl _abort, r31
    dispose 0x00000000, 0x00000001, [r31]
_func1:
    .stack _func1 = 0
    movhi HIGHW1(#_glb), r0, r2
    ld.w LOWW(#_glb)[r2], r5
    add 0x00000001, r5
    st.w r5, LOWW(#_glb)[r2]
    jmp [r31]
_func2:
    .stack _func2 = 0
    movhi HIGHW1(#_glb), r0, r2
    ld.w LOWW(#_glb)[r2], r5
    add 0xFFFFFFFF, r5
    st.w r5, LOWW(#_glb)[r2]
    jmp [r31]
_main:
    .stack _main = 8
    prepare 0x00000041, 0x00000000
    movhi HIGHW1(#_pf), r0, r20
    ld.w LOWW(#_pf)[r20], r20
    mov r20, r6
    jarl ___control_flow_integrity, r31 ; Call the checking function.
    jarl [r20], r31 ; Indirect call of the function func1.
    jarl _func2, r31 ; Direct call of the function func2.
    dispose 0x00000000, 0x00000041, [r31]
    .section .bss, bss
    .align 4
_glb:
    .ds (4)
    .section .data, data
    .align 4
_pf:
    .dw #_func1
    .section .const, const
```

-pic [V1.07.00 or later]

This option enables the PIC facility.

[Specification format]

```
-pic
```

- Interpretation when omitted
  The PIC facility is disabled.

[Detailed description]

- The PIC facility makes the section to which functions are allocated position-independent.
  See "4.2.6.1 Allocation of function and data to section " and "8.6 PIC/PID Facility" for details about the PIC facility.
- When this option is specified, the section to which the function code is output is changed from the text attribute section to the pctext attribute section.
- Referencing a function allocated to the pctext attribute section is always performed in PC-relative mode. This allows the pctext attribute section to be allocated at a desired address after linkage.
- When this option is specified, the predefined macro __PIC will be valid.
- If this option is not specified simultaneously with the -pirod option, an error will occur.

-pirod [V1.07.00 or later]

This option enables the PIROD facility.

[Specification format]

```
-pirod
```

- Interpretation when omitted
  The PIROD facility is disabled.

[Detailed description]

- The PIROD facility makes the section to which constant data, such as const variables or string literal, is allocated position-independent.
  See "4.2.6.1 Allocation of function and data to section" and "8.6 PIC/PID Facility" for details about the PIROD facility.
- When this option is specified, the section to which constant data is output is changed from the const attribute section to the pcconst32 attribute section.
- Referencing constant data allocated to the pcconst32 attribute section is always performed in PC-relative mode. This allows the pcconst32 attribute section to be allocated at a desired address after linkage.
- When this option is specified, the predefined macro __PIROD will be valid.
- If this option is not specified simultaneously with the -pic option, an error will occur.
- If this option is specified simultaneously with the -Omap or -Osmap option, an error will occur.

-pid [V1.07.00 or later]

This option enables the PID facility.

[Specification format]

```
-pid
```

- Interpretation when omitted
  The PID facility is disabled.

[Detailed description]

- The PID facility makes the section to which variable data is allocated position-independent.
  See "4.2.6.1 Allocation of function and data to section" and "8.6 PIC/PID Facility" for details about the PID facility.
- When this option is specified, the section to which variable data is output is changed from the data or bss attribute section to the sdata32 or sbss32 attribute section, respectively.
- Referencing variable data allocated to the sdata32 or sbss32 attribute section is always performed in GP-relative mode. This allows the sdata32 or sbss32 attribute section to be allocated at a desired address after linkage.
- When this option is specified, the predefined macro __PID will be valid.
- If this option is specified simultaneously with the -r4=none option, an error will occur.
- If this option is specified simultaneously with the -Omap or -Osmap option, an error will occur.

Information file output control

The information file output control option is as follows.

- -Xcref

-Xcref

This option outputs the static analysis information file.

[Specification format]

```
-Xcref=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:

```
>ccrh -Xcref=info.cref -Xcommon=rh850 main.cs
```

Error output control

The error output control options are as follows.

- -Xerror_file

-Xerror_file

This option outputs error messages to a file.

[Specification format]

```
-Xerror_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:

```
>ccrh -Xerror_file=err -Xcommon=rh850 main.c
```

Warning message output control

The warning message output control options are as follows.

- -Xno_warning
- -change_message [V1.07.00 or later]

-Xno_warning

This option suppresses outputting warning messages of the specified number.

[Specification format]

```
-Xno_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.
- When this option is specified more than once, all specifications will be valid.
- The error number specified by this option is the rightmost 5 digits of the 7-digit number following the "W".
  See "10. MESSAGE" for error numbers.
- A message whose type was changed to error level by the -change_message option cannot be controlled by this option.
- The message numbers that can be controlled by this option are as follows:
  - W0520000 to W0529999 and W0550000 to W0559999 [V1.06.00 or earlier]
  - W0510000 to W0559999 [V1.07.00 or later]

[Example of use]

- To suppress outputting warning message "W0520111" describe as:

```
>ccrh -Xno_warning=20111 -Xcommon=rh850 main.c
```

-change_message [V1.07.00 or later]

This option changes specified warning messages into error messages.

[Specification format]

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

[Detailed description]

- This option changes specified warning messages into error messages.
  The message numbers that are targeted by this option are W0510000 to W0549999.
- Specify the rightmost 5 digits of the message number as *num*, *num1*, and *num2*.
  If *num1-num2* is specified, it is assumed that error numbers within the range have been specified.
- If *num* or *num1-num2* is omitted, all target warning messages are changed into error messages.
- When this option is specified more than once, all specifications will be valid.
- If a message number that does not exist is specified, the specification is ignored.
  For message numbers, see "10. MESSAGE".

[Example of use]

- To change W0520000 to W0549999 into error messages, describe as:

```
>ccrh -change_message=error=20000-49999 a.c
```

Phase individual option specification

The phase individual option specification options are as follows.

- -Xasm_option
- -Xlk_option

-Xasm_option

This option specifies assemble options.

[Specification format]

```
-Xasm_option=arg
```

- Interpretation when omitted
  The ccrh driver interprets all specified options.

[Detailed description]

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

[Example of use]

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

```
>ccrh -Xasm_option=-Xprn_path -Xcommon=rh850 main.c
```

The example above has the same meaning as the following.

```
>ccrh -S -Xcommon=rh850 main.c
>asrh -Xprn_path -Xcommon=rh850 main.asm
```

-Xlk_option

This option specifies link options.

[Specification format]

```
-Xlk_option=arg
```

- Interpretation when omitted
  The ccrh driver interprets all specified options.

[Detailed description]

- This option passes *arg* to the linker as the link option.
- Use this option to pass a file to the linker containing an identifier that the ccrh driver does not recognize as input to the linker.
- An error will occur if *arg* is a non-existent link option.
- An error will occur if *arg* is omitted.

[Example of use]

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

```
>ccrh -Xlk_option=-form=relocate -Xcommon=rh850 main.c
```

The example above has the same meaning as the following.

```
>ccrh -c -Xcommon=rh850 main.c
>rlink -form=relocate main.obj
```

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:

```
>ccrh @command.txt -Xcommon=rh850
```

2.5.2 Assemble 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, 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 (").
- When the -Xprn_path or -Xasm_far_jump option is specified for ccrh command, the -Xasm_option option must be used.

The types and explanations for options are shown below.

Table 2.3 Assemble Options

Classification	Option	Description
Version/help display specification	-V	This option displays the version information of asrh.
	-h	This option displays the descriptions of asrh options.
Output file specification	-o	This option specifies the output file name.
	-Xobj_path	This option specifies the folder to save an object file generated during assembling.
	-Xprn_path	This option specifies the folder to save the assemble list file.
Source debugging control	-g	This option outputs information for source debugging.
Device specification	-Xcommon	This option specifies that an object file common to the various devices is generated.
	-Xcpu	This option specifies that an object for the specified core is generated.
Optimization	-goptimize [V2.01.00 or later]	This option generates information for link-time optimization.
Symbol definition specification	-D	This option defines assembler symbols.
	-U	This option deletes the assembler symbol definition by the -D option.
Include file reading path specification	-I	This option specifies the folder to search include files.
Japanese/Chinese character control	-Xcharacter_set	This option specifies the Japanese/Chinese character code.

Classification	Option	Description
Generated code control	-Xreg_mode	This option specifies the register mode.
	-Xreserve_r2	This option reserves the r2 register.
	-Xep	This option specifies how to handle the ep register.
	-pic [V1.07.00 or later]	This option enables the PIC facility.
	-pirod [V1.07.00 or later]	This option enables the PIROD facility.
	-pid [V1.07.00 or later]	This option enables the PID facility.
Assembler control specification	-Xasm_far_jump	This option controls outputting far jump for an assembly source file.
Error output control	-Xerror_file	This option outputs error messages to a file.
Warning message output control	-Xno_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
- -h

-V

This option displays the version information of asrh.

[Specification format]

```
-V
```

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

[Detailed description]

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

[Example of use]

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

```
>asrh -V -Xcommon=rh850
```

-h

This option displays the descriptions of asrh options.

[Specification format]

```
-h
```

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

[Detailed description]

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

[Example of use]

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

```
>asrh -h -Xcommon=rh850
```

Output file specification

The output file specification options are as follows.

- -o
- -Xobj_path
- -Xprn_path

-o

This option specifies the output file name.

[Specification format]

```
-ofile
```

- 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 if this option is specified, when a error occurs, the object file cannot 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:

```
>asrh -osample.obj -Xcommon=rh850 main.asm
```

-Xobj_path

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

[Specification format]

```
-Xobj_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 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".
- 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 assembling to folder "D:\sample", describe as:

```
>asrh -Xobj_path=D:\sample -Xcommon=rh850 main.asm
```

-Xprn_path

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

[Specification format]

```
-Xprn_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 under the source file name with the extension replaced by ".prn" to *path*.
  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 under the source file name with the extension replaced by ".prn".

[Example of use]

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

```
>asrh -Xprn_path=D:\sample -Xcommon=rh850 main.asm
```

Source debugging control

The source debugging control options are as follows.

- -g

-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.

[Example of use]

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

```
>asrh -g -Xcommon=rh850 main.asm
```

Device specification

The device specification options are as follows.

- -Xcommon
- -Xcpu

-Xcommon

This option specifies that an object file common to the various devices is generated.

[Specification format]

```
-Xcommon=series
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies that an object file common to the various devices is generated.
- This option is invalid in V2.00.00 or later versions. If this option is specified, it will be ignored but no error will occur due to the compatibility with conventional versions. No warning is output in this case.
- v850e3v5 or rh850 can be specified for *series*.
- An error will occur in any of the following cases.
  - When *series* is omitted
  - When a parameter that is not specifiable is specified for *series*
  - When this option is omitted [V1.01.00 or earlier]

[Remark]

This option does not affect the output code.
When selecting the instruction set to be used, specify the -Xcpu option.

-Xcpu

This option specifies that an object for the specified core is generated.

[Specification format]

```
-Xcpu=core
```

- Interpretation when omitted
  An object for G3M is generated.

[Detailed description]

- This option specifies that an object for core *core* is generated.
- The items that can be specified as *core* are shown below.

g3m	Generates an object for G3M.
g3k	Generates an object for G3K.
g3mh	Generates an object for G3MH. [V1.02.00 or later]
g3kh	Generates an object for G3KH. [V1.03.00 or later]
g4mh	Generates an object for G4MH. [V2.00.00 or later]

- If this option is specified more than once, the last specification is valid.
- An error will occur in any of the following cases.
  - When the parameter is omitted
  - When a parameter that is not specifiable is specified

Optimization

The Optimization options are as follows.

- -goptimize [V2.01.00 or later]

-goptimize [V2.01.00 or later]

This option generates information for link-time optimization.

[Specification format]

```
-goptimize
```

- Interpretation when omitted
  None

[Detailed description]

- This option generates additional information used at link-time optimization in the output file.
- When this option is specified for a file, link-time optimization will be applied at linkage of the file.
  For details on link-time optimization, see the description of the link option -OPtimize.

Symbol definition specification

The symbol definition specification options are as follows.

- -D
- -U

-D

This option defines assembler symbols.

[Specification format]

```
-Dname[=def][name[=def]]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option defines *name* as an assembler symbol.
- 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 a positive (+) sign) 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 -U option are specified for the same assembler symbol, the option specified last will be valid.

[Example of use]

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

```
>asrh -Dsample=256 -Xcommon=rh850 main.asm
```

-U

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

[Specification format]

```
-Uname[,name]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option deletes the definition of assembler symbol *name* by the -D option.
- An error will occur if *name* is omitted.
- This option cannot delete the definition by describing "*name* .SET *def*".
- This option can be specified more than once.
- If both this option and -D option are specified for the same assembler symbol, the option specified last will be valid.

[Example of use]

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

```
>asrh -Utest -Xcommon=rh850 main.asm
```

Include file reading path specification

The include file reading path specification options are as follows.

- -I

-I

This option specifies the folder to search include files.

[Specification format]

```
-Ipath[,path]...
```

- Interpretation when omitted
  The include file is searched from the standard include file folder.

[Detailed description]

- This option specifies the folder to search include files that are read by assembler control instruction "$INCLUDE/ $BINCLUDE" as *path*.
  Include files are searched according to the following sequence.
  (1) 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).)
  (2) Folder with source file
  (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 include files from folder "D:\include", "D:\src", and the current folder in that order, describe as:

```
>asrh -ID:\include -Xcommon=rh850 D:\src\main.asm
```

Japanese/Chinese character control

The Japanese/Chinese character control option is as follows.

- -Xcharacter_set

-Xcharacter_set

This option specifies the Japanese/Chinese character code.

[Specification format]

```
-Xcharacter_set=code
```

- 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 source file.
- The items that can be specified as *code* 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 source 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 *code* 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:

```
>asrh -Xcharacter_set=euc_jp -Xcommon=rh850 main.asm
```

Generated code control

The generated code control options are as follows.

- -Xreg_mode
- -Xreserve_r2
- -Xep
- -pic [V1.07.00 or later]
- -pirod [V1.07.00 or later]
- -pid [V1.07.00 or later]

-Xreg_mode

This option specifies the register mode.

[Specification format]

```
-Xreg_mode=mode
```

- Interpretation when omitted
  The 32-register mode object file is generated.

[Detailed description]

- This option generates the object file for the specified register mode.
- This option limits the number of registers used by ccrh to 32 (the 32-register mode) or 22 (the 22-register mode or register mode "common") and embeds the magic number into the object file.
- Use register mode "common" to generate the object file that does not depend on register modes.
- The items that can be specified as *mode* are shown below.
  An error will occur if any other item is specified.

Register Mode (*mode*)	Working Registers	Registers for Register Variables
common	r10 to r14	r25 to r29
22	r10 to r14	r25 to r29
32	r10 to r19	r20 to r29

- An error will occur if *mode* is omitted.
- If 32-register mode object files and 22-register mode object files are mixed, an error will occur at linkage.

[Example of use]

- To generate the 22-register mode object file, describe as:

```
>asrh -Xreg_mode=22 -Xcommon=rh850 main.asm
```

-Xreserve_r2

This option reserves the r2 register.

[Specification format]

```
-Xreserve_r2
```

- Interpretation when omitted
  The compiler uses the r2 register without reserving it.

[Detailed description]

- This option reserves the r2 register and generates code that does not use this register by the compiler.

[Example of use]

- To reserve the r2 register and generates code that does not use this register by the compiler, describe as:

```
>asrh -Xreserve_r2 -Xcommon=rh850 main.asm
```

-Xep

This option specifies how to handle the ep register.

[Specification format]

```
-Xep=mode
```

- Interpretation when omitted
  The ep register is treated as a register guaranteeing the value before and after the function call.

[Detailed description]

- This option specifies how to handle the ep register.
- The items that can be specified as *mode* are shown below.
  An error will occur if any other item is specified.

fix	Fixes the value of the ep register for the entire project. Specify this parameter when EP-relative sections in the project are used.
callee	Treats the ep register as a register guaranteeing the value before and after the function call. Specify this parameter when the -Omap or -Osmap option is specified.

- An error will occur if *mode* is omitted.
- The same specification must be made for all source files. A different specification cannot be made for each source file. If there are object files with different specifications, an error will occur at linkage.

[Example of use]

- To fix the value of the ep register for the entire project, describe as:

```
>asrh -Xep=fix main.asm
```

-pic [V1.07.00 or later]

This option enables the PIC facility.

[Specification format]

```
-pic
```

- Interpretation when omitted
  The PIC facility is disabled.

[Detailed description]

- When this option is specified, the relocation attribute that can be specified in the .cseg or .section directive is changed.
  If a relocation attribute shown below is specified, an error will occur.
  When this option is specified: TEXT
  When this option is not specified: PCTEXT
- When this option is specified, the predefined macro __PIC will be valid.
- If this option is not specified simultaneously with the -pirod option, an error will occur.
- This option only controls the relocation attribute that can be specified. The processing for determining whether there is an error in the code for referencing functions is not performed.

-pirod [V1.07.00 or later]

This option enables the PIROD facility.

[Specification format]

```
-pirod
```

- Interpretation when omitted
  The PIROD facility is disabled.

[Detailed description]

- When this option is specified, the relocation attribute that can be specified in the .cseg or .section directive is changed.
  If a relocation attribute shown below is specified, an error will occur.
  When this option is specified: CONST, ZCONST, or ZCONST23
  When this option is not specified: PCCONST16, PCCONST23, or PCCONST32
- When this option is specified, the predefined macro __PIROD will be valid.
- If this option is not specified simultaneously with the -pic option, an error will occur.
- This option only controls the relocation attribute that can be specified. The processing for determining whether there is an error in the code for referencing constant data is not performed.

-pid [V1.07.00 or later]

This option enables the PID facility.

[Specification format]

```
-pid
```

- Interpretation when omitted
  The PID facility is disabled.

[Detailed description]

- When this option is specified, the relocation attribute that can be specified in the .dseg or .section directive is changed.
  If a relocation attribute shown below is specified, an error will occur.
  When this option is specified: DATA, ZDATA, ZDATA23, BSS, ZBSS, or ZBSS23
  When this option is not specified: SDATA32, SBSS32, EDATA32, or EBSS32
- When this option is specified, the predefined macro __PID will be valid.
- This option only controls the relocation attribute that can be specified. The processing for determining whether there is an error in the code for referencing data is not performed.

Assembler control specification

The assembler control specification option is as follows.

- -Xasm_far_jump

-Xasm_far_jump

This option controls outputting far jump for an assembly source file.

[Specification format]

```
-Xasm_far_jump
```

- Interpretation when omitted
  Assembly is performed as a jarl or jr instruction.

[Detailed description]

- For an assembly source file, this option assumes that all jarl and jr instructions described in the source are jarl32 and jr32 instructions, and assembling is performed.
- If you wish to control individual instructions, add jarl22/jarl32 or jr22/jarl32 to the source.
- This option does not affect the jump instruction.
- If this option is specified for a C source file, that will be ignored without outputting a warning.

[Example of use]

- To assume that all jarl and jr instructions described in the source are jarl32 and jr32 instructions, and perform assembling, describe as:

```
>asrh -Xasm_far_jump -Xcommon=rh850 main.asm
```

Error output control

The error output control option is as follows.

- -Xerror_file

-Xerror_file

This option outputs error messages to a file.

[Specification format]

```
-Xerror_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:

```
>asrh -Xerror_file=err -Xcommon=rh850 main.asm
```

Warning message output control

The warning message output control options are as follows.

- -Xno_warning

-Xno_warning

This option suppresses outputting warning messages of the specified number.

[Specification format]

```
-Xno_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".
  See "CS+ Integrated Development Environment User's Manual: Message" for error numbers.
- This option controls the warning messages of the compiler and assembler and does not control the messages of the optimizing linker.
- 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 "W0550002" and "W0550003", describe as:

```
>asrh -Xno_waning=50002,50003 -Xcommon=rh850 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:

```
>asrh @command.txt -Xcommon=rh850
```

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 ccrh command, the -Xlk_option option must be used.

The types and explanations for options are shown below.

Table 2.4 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 [V2.02.00 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 [V1.06.00 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.
	-MAp	This option outputs the external variable allocation information 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 user of the external defined symbol that is not referenced.
	-BYte_count	This option specifies the maximum byte count for a data record.
	-FIX_RECORD_LENGTH_AND_ALIGN [V1.07.00 or later]	Fixes the format of data records to be output.
	-PADDING	This option fills in data at the end of a section.
	-OVERRUN_FETCH	This option prevents reading of vacant areas due to overrun fetch.
	-RESERVE_PREFETCH_AREA [V2.04.01 or later]	This option generates and reserves a section in an area that can be prefetched.
	-CRc	This option outputs the CRC code.
	-CFI [Professional Edition only] [V1.07.00 or later]	Generates the function list for use in detecting illegal indirect function calls.
	-CFI_ADD_Func [Professional Edition only] [V1.07.00 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] [V1.07.00 or later]	Specifies modules which are to be exempted from the function list for use in detecting illegal indirect function calls.
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 / -NOOPtimize [V2.01.00 or later]	This option specifies whether link-time optimization is to be executed.
	-SEction_forbid [V2.01.00 or later]	This option suppresses link-time optimization of specific sections.
	-Absolute_forbid [V2.01.00 or later]	This option suppresses link-time optimization in a specific address range.
	-SYmbol_forbid [V2.01.00 or later]	This option suppresses link-time optimization of specific symbols.
	-ALLOW_OPTIMIZE_ENTRY_BLOCK [V2.06.00 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.
	-ALIGNED_SECTION	This option changes the number of alignment of the section to 16 bytes.
Verify specification	-CPu	This option checks the consistency of the address to which the section is allocated.
Subcommand file specification	-SUbcommand	This option specifies options with a subcommand file.

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 [V2.01.00 or later]	This option changes a symbol name or section name 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 [V2.03.00 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 [V2.02.00 or later]

-Input

This option specifies the input file.

[Specification format]

```
-Input=suboption [{,| Δ } ...]
  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.
- 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.

[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.

[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 link-time optimization (-optimize[=symbol_delete]). If *address* is specified with this option, link-time optimization 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 [V2.02.00 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 [V1.06.00 or later]
- -ROm
- -OUtput
- -MAp
- -SPace
- -Message
- -NOMessage
- -MSg_unused
- -BYte_count
- -FIX_RECORD_LENGTH_AND_ALIGN [V1.07.00 or later]
- -PADDING
- -OVERRUN_FETCH
- -RESERVE_PREFETCH_AREA [V2.04.01 or later]
- -CRc
- -CFI [Professional Edition only] [V1.07.00 or later]
- -CFI_ADD_Func [Professional Edition only] [V1.07.00 or later]
- -CFI_IGNORE_Module [Professional Edition only] [V1.07.00 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 2.5 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, -map, -padding, -hide, -optimize/-nooptimize, -absolute_forbid, -symbol_forbid, -section_forbid, -compress, -nocompress, -rename, -lib_rename, -delete, -define, -fsymbol, -stack, -memory, -msg_unused, -show={all|symbol|reference|xreference|total_size|struct|relocation_attribute|cfi}, -aligned_section, -overrun_fetch, -cfi, -cfi_add_func, -cfi_ignore_module

Output Format	Specified Option	File Format That Can Be Input	Specifiable Option Note 1
Relocate	-extract specified	Library file	-library, -output
	Other than above	Object file Relocatable file Binary file Library file	-input, -library, -binary, -debug, -nodebug, -output, -hide, -rename, -lib_rename, -delete, -show={all|symbol|xreference|total_size}
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, -map, -space, -optimize/-nooptimize, -absolute_forbid, -symbol_forbid, -section_forbid, -rename, -lib_rename, -delete, -define, -fsymbol, -stack, -record Note 2, -end_record Note 2, -s9 Note 2, -byte_count Note 3, -fix_record_length_and_align Note7, -padding, -memory, -msg_unused, -show={all|symbol|reference|xrefer-ence|total_size|struct|relocation_attribute|cfi}, -aligned_section, -overrun_fetch Note 4, -crc, -cfi, -cfi_add_func, -cfi_ignore_module
		Load module file	-input, -output, -record Note 2, -end_record Note 2, -s9 Note 2, -byte_count Note 3, -fix_record_length_and_align Note 7, -show={all|symbol|reference|xreference}, -crc
		Intel HEX file Note 5	-input, -output
		Motorola S-record file Note 5	-input, -output, -s9 Note 2
Library	-strip specified	Library file	-library, -output, -memory Note 6, -show
	-extract specified	Library file	-library, -output
	Other than above	Object file Relocatable file	-input, -library, -output, -hide, -rename, -delete, -replace, -memory Note 6, -show={all|symbol|section}, -allow_duplicate_module_name

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

Note 2. 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. The -overrun_fetch option is valid only when the -form=hexadecimal or -form=stype option is specified.

Note 5. 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 6. The -memory option cannot be specified when the -hide option is specified.

Note 7. The -fix_record_length_and_align 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 [V1.06.00 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*. [V2.06.00 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/8000
```

[Remark]

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

-OUtput

This option specifies the output file.

[Specification format]

```
-OUtput=suboption[, ...]
  suboption := file
             |  file=range
             |  file=/load-address
             |  file=range/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 *load-address* is specified, the first load address in the output file will be changed to the value specified with *load-address* when an Intel Hex file or Motorola S-record file is output. [V2.00.00 or later]
- If this option and the -form={absolute|hexadecimal|stype|binary} option are specified at the same time, two or more files can be specified.

[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]

- *load-address* can be specified when form={hexadecimal|stype} has been specified.
- 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").

-MAp

This option outputs the external variable allocation information file.

[Specification format]

```
-MAp[=file]
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs external variable allocation information file *file* that is used by the compiler in optimizing access to external variables.
- If the specification of the file name is omitted, the file name is the one specified by the -output option or "*first-input file-name*.bls".
- If the order of the declaration of variables in the external variable allocation information file is not the same as the order of the declaration of variables found when the object was read after recompilation, an error will occur.
- In the following case, the linker outputs the external variable allocation information file and, when the –LIst option is specified, outputs the list file. After that, the linker terminates operation normally. Note that the linker does not output a load module file in this case. [V1.05.00 or later]
  - When the section allocation address exceeds the allowable address range:
    In the external variable allocation information file, information regarding only the symbols and sections allocated within the allowable areas are output.

[Example of use]

- To output external variable allocation information file file.bls, describe as:

```
>rlink a.obj b.obj -output=c.abs -map=file.bls
```

[Remark]

- This option is valid only when the -form={absolute|hexadecimal|stype|binary} option is specified.

-SPace

This option fills the vacant area of 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 value is output to vacant areas between the specified sections.
  - When the -Output option is used to specify a range of addresses as the range for output:
    The specified value is output to vacant areas within the specified range.
  - When the -FIX_RECORD_LENGTH_AND_ALIGN option is specified:
    - The specified 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 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 "ffH" within the range from address 100H to address 2FFH, describe as:

```
>rlink a.obj b.obj -form=hexadecimal -output=file1=100-2ff -start=P1/100,P2/200
-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 information messages.

[Specification format]

```
-Message
```

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

[Detailed description]

- This option output information messages.

[Example of use]

- To output information messages, 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

This option notifies the user of the external defined symbol that is not referenced.

[Specification format]

```
-MSg_unused
```

- Interpretation when omitted
  None

[Detailed description]

- This option notifies the user of the external defined symbol that is not referenced during link processing through an output message.

[Example of use]

- To notify the user of the external defined symbol that is not referenced, describe as:

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

[Remark]

- 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.
- If there are references to constant symbols within the same file, references are not correctly analyzed so that information notified by the output messages will be incorrect.

-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]

- If the -form=hexadecimal option is not specified, this option will be invalid. [V1.06.00 or earlier]
- If the -form={hexadecimal|stype} option is not specified, this option will be invalid. [V1.07.00 or later]

-FIX_RECORD_LENGTH_AND_ALIGN [V1.07.00 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, 0 (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]

- If 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.

[Example of use]

- In the following case, 2 bytes of padding data are filled in the .const section, and linking is performed with a size of 0x08.
  Alignment of the .const section: 4 bytes
  Size of the .const section: 0x06 bytes
  Alignment of the .text section: 2 byte
  Size of the .text section: 0x02 bytes

```
>rlink a.obj b.obj -start=.const,.text/0 -padding
```

- In the following case, if 2 bytes of padding data are filled in the .const section, and linking is performed with a size of 0x08, then an error will be output because it overlaps with the .text section.
  Alignment of the .const section: 4 bytes
  Size of the .const section: 0x06 bytes
  Alignment of the .text section: 2 byte
  Size of the .text section: 0x02 bytes

```
>rlink a.obj b.obj -start=.const/0,.text/6 -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.
- Padding is performed only in the sections for text data, const variables, and initialized variables in V1.00.01. In V1.01.00 and later versions, however, padding is also done in the sections for uninitialized variables.

-OVERRUN_FETCH

This option prevents reading of uninitialized areas due to overrun fetch.

[Specification format]

```
-OVERRUN_FETCH
```

- Interpretation when omitted
  None

[Detailed description]

- When a 128-byte or larger vacant area exists immediately after a ROM section, NOP instructions for 128 bytes are inserted in the vacant area because the uninitialized code flash area may be prefetched when that ROM section is read from.

  Note The amount of 128 bytes is a rough indication for the size independent of the prefetch size of each microcontroller. If the vacant area between a ROM section and its subsequent section is less than 128 bytes, NOP instructions are not inserted because an uninitialized area will not exist due to the fact that the unit for programming the code flash area is 256 bytes.

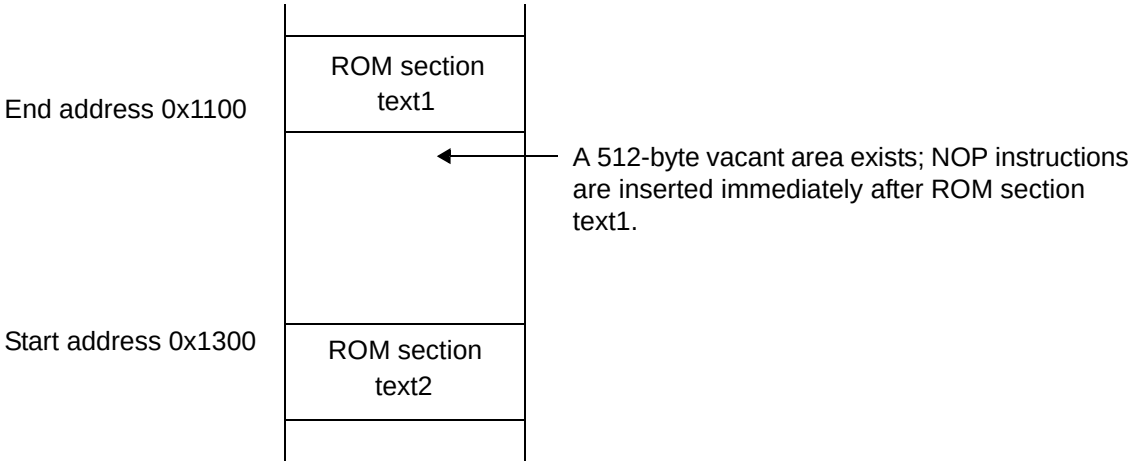
  NOP instructions are inserted according to the allocation specified by the -start option and output with the following section name obtained from the name of the section immediately before the inserted NOP instructions.

```
$sss_fetch??
```

  sss: Name of the section immediately before the inserted NOP instructions
  ??: 0199

(1) Insert between sections

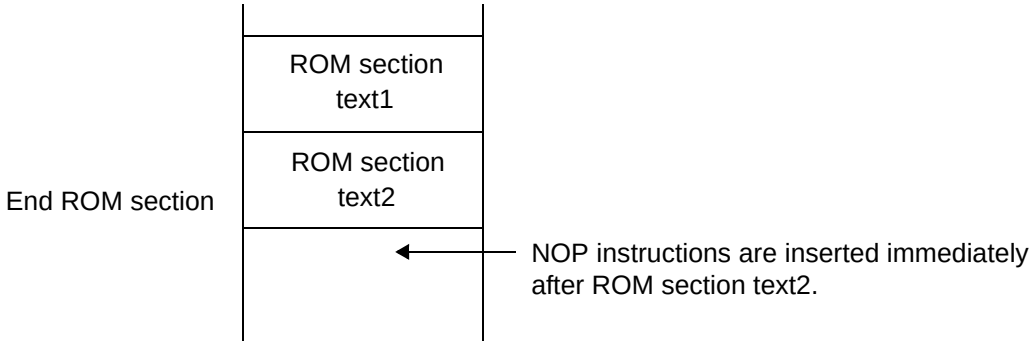
- NOP instructions are inserted when a 128-byte or larger vacant area exists between two ROM sections (executable program sections or sections (except program sections) allocated to the ROM area) or between a ROM section and a RAM section (a section allocated to the RAM area).



- For insertion between a ROM section and RAM section, NOP instructions are inserted only when the RAM section is allocated to larger addresses than the ROM section addresses (When the RAM section addresses are smaller than the ROM section addresses, NOP instructions are not inserted between them).

(2) Insert immediately after the end section

- NOP instructions are inserted immediately after the end ROM section.



- When an address range is specified by the -cpu option, if the vacant area between the end address of the end section and the end address of the specified range is smaller than 128 bytes, NOP instructions will not be inserted.
- If the vacant area between the end address of the end section and an 8-Mbyte address boundary is smaller than 128 bytes, NOP instructions will not be inserted.

[Example of use]

- To prevent reading of vacant areas due to overrun fetch, describe as:

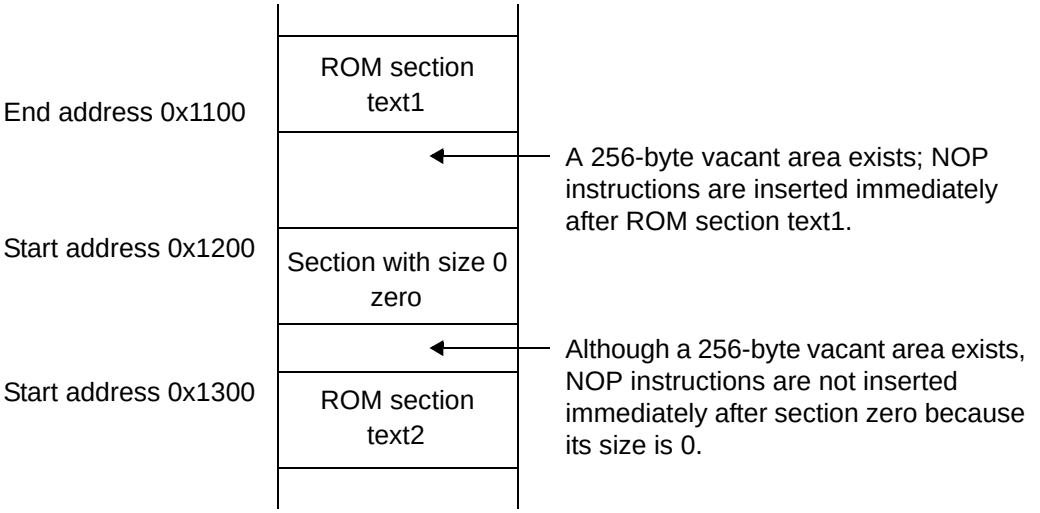
```
>rlink a.obj b.obj -overrun_fetch
```

[Remark]

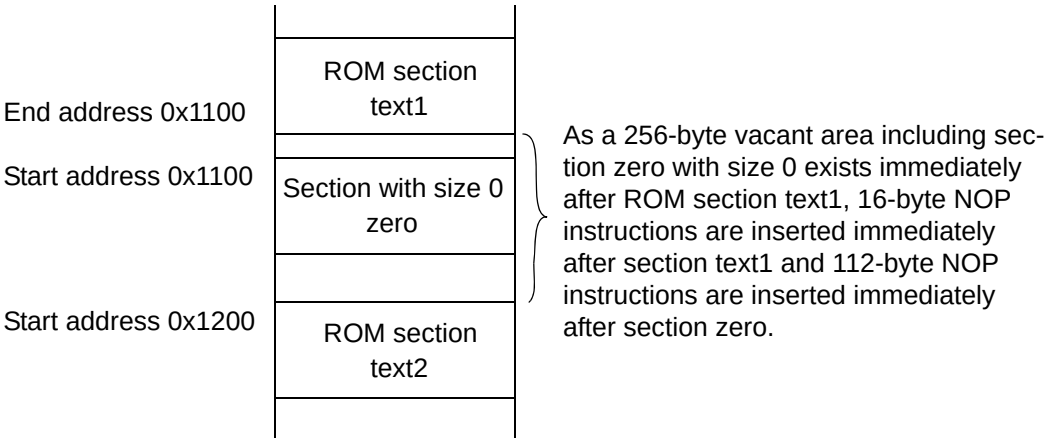
- When no address range is specified by the -cpu option and the end section is a ROM section, NOP instructions are always inserted immediately after the end ROM section.
- If a load module file is input, this option will be invalid.
- If the -form={object|relocate|library} option is specified, this option will be invalid.
- When overlaid sections are specified by the -start option, NOP instructions are not inserted for a string of sections specified as being overlaid by the -start option.

Example 1. When option -start=A,B,E/400,C,D:F:G/8000 is specified, NOP instructions are not inserted between sections C, D, F, and G and after the end code section because overlaid sections are specified.

Example 2. When a 128-byte or larger vacant area exists immediately after a section with size 0, NOP instructions are not inserted.



Example 3. NOP instructions are inserted when a 128-byte or larger vacant area including a section with size 0 exists immediately after a ROM section.



-RESERVE_PREFETCH_AREA [V2.04.01 or later]

This option generates and reserves a section in an area that can be prefetched.

[Specification format]

```
-RESERVE_PREFETCH_AREA[=section]
```

- Interpretation when omitted
  No section is generated.

[Detailed description]

- This option generates and reserves a section in an area that can be prefetched.
- An area for 16 bytes following the end of a section that can be accessed for the prefetch operation of the processor is reserved as an NOP instruction section.
- The section is output with the following section name obtained from the name of the section immediately before the inserted NOP instruction.

  Output section name

  ```
  $sss_fetch??
  ```

  sss : Name of the section immediately before the inserted NOP instruction
  ?? : 01 to 99
- If an area for the section cannot be reserved, an error occurs.

[Remark]

- If the input file has the absolute format, this option outputs an error message and terminates execution.
- If form={object | relocate | library | binary} is specified, this option outputs an error message and terminates execution.
- When overlaid sections are specified by the start option, NOP instructions are not inserted for a string of sections specified as being overlaid by the start option.

Example 1. -start=A,B,E/400,C,D:F:G/8000
NOP instructions are not inserted between sections C, D, F, and G and after the end code section because overlaid sections are specified.

- If a section with size 0 is allocated immediately after the program section, the NOP instruction is inserted immediately after that section.

-CRc

This option outputs the CRC code.

[Specification format]

```
-CRc=output_address=operation_range[/operation_method][(initial_value)][:endian]
    operation_range  := { start_address-end_address | section }[,...]
    endian           := { BIG | LITTLE}[-size-offset ]
```

- Interpretation when omitted
  None

[Detailed description]

- CRC (cyclic redundancy check) operation is done for the specified range of section data in the order from the lower to the higher addresses, and the operation result is output to the specified output address in the specified endian mode.
- Specify one of the following as the operation method. If the specification of the operation method is omitted, operation is performed assuming that 32-ETHERNET has been specified.

Operation Method	Description
CCITT	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	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	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	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	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 $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$.

- The range of values that can be specified for *initial-value* 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 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.

- The operation result is output to the specified output address by writing at the offset location from the beginning of the area allocated by size in the byte order specified with BIG or LITTLE. 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 endian is omitted, the result is written in little endian.
- When the space option is not specified, space=FF 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. [V2.05.00 or later]

[Example of use]

- To perform CRC operation for the area from 0x1000 to 0x2FFD and write the result to address 0x2FFE, describe as:

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

- To output the result of CRC operation performed for the area from 0x1000 to 0x1FFF to address 0x2FFC and the result of CRC operation performed for the area from 0x2000 to 0x2FFB to address 0x2FFE, describe as:

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

[Remark]

- When multiple load module files are input, the compiler outputs a warning message and ignores this option.
- This option is valid in any of the following cases. For any other cases, the error message is output and execution is terminated.
  - When -form={hexadecimal|stype} is specified [V1.07.00 or earlier]
  - When -form={hexadecimal|stype|bin} is specified [V2.00.00 or later]
- 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 to terminate execution if the CRC operation range includes an overlaid area.
- The following can be specified for the size and offset when specifying the endian. For any other cases, the error message is output and execution is terminated.
  - LITTLE
  - LITTLE-2-0
  - LITTLE-4-0
  - BIG
  - BIG-2-0
  - BIG-4-0

-CFI [Professional Edition only] [V1.07.00 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] [V1.07.00 or later]' compile option.
  Since the linker generates the function list for the .const section, the .const 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] [V1.07.00 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] [V1.07.00 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] [V1.07.00 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 [, ...]
   suboption := file
              |  file ( module [, ... ] )
```

- Interpretation when omitted
  None

[Detailed description]

- This option specifies object files or library files ([V2.00.00 or later]) to be exempted from 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] [V1.07.00 or later]' compiler option.
- When a library file is specified, a module name within the library can be specified.
- If the specified file does not exist, an error will occur.
- If this option is specified more than once, the functions of all specified 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
```

- To remove functions in the c module in the b.lib library from the function list, code as: [V2.00.00 or later]

```
>rlink -cfi -cfi_ignore_module=b.lib(c) -lib=b.lib a.obj
```

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.

- When this option is specified together with the –MAp option, this option outputs the link map information and symbol information even if the section allocation address exceeds the allowable address range. In this case, "**OVER**" is output. [V1.05.00 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 contents.
Reference	Not specifiable	Outputs the symbol address, size, type, optimization contents, 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.
STRUCT	Not specifiable	Outputs in the symbol information the structure and union member information defined in files that have been compiled with -g specified (this setting will be invalid if -form=rel or obj is specified).
RELOCATION_ATTRIBUTE [V1.06.00 or later]	Not specifiable	Outputs the relocation attribute.
CFI [V1.07.00 or later]	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.
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,struct option is specified. When the -form=hexadecimal/stype/binary option Outputs the same information as when the -show=symbol,reference,xreference,total_size,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	Refer-ence	Section	Xrefer-ence	Total_size	STRUCT	RELOCATION_ATTRIBUTE	CFI
-form=absolute	only -show	Valid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Valid	Invalid	Valid	Valid	Valid-Note 4	Invalid	Invalid
-form=library	only -show	Valid	Invalid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Invalid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
-form=relocate	only -show	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Invalid	Invalid	Valid-Note 1	Valid	Invalid	Invalid	Invalid
-form=object	only -show	Valid	Valid	Invalid	Invalid	Invalid	Invalid	Invalid	Invalid
	-show=all	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
	-show=all	Valid	Valid	Invalid	Valid	Valid-Note 1	Valid-Note 4	Invalid	Invalid

Note 1. If a load module file is input, this combination will be invalid.

Note 2. If a input file is an Intel Hex file, the -show option cannot be specified.

Note 3. If a input file is a Motorola S-record file, the -show option cannot be specified.

Note 4. If the -rename, -lib_rename, -hide, or -compress option or optimization by deleting unreferenced symbols (-optimize=symbol_delete) is specified, this combination will be invalid.

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.
- Both -show=total_size option and -total_size option output the same information.

Optimization

The Optimization options are as follows.

- -OPtimize / -NOOPtimize [V2.01.00 or later]
- -SEction_forbid [V2.01.00 or later]
- -Absolute_forbid [V2.01.00 or later]
- -SYmbol_forbid [V2.01.00 or later]
- -ALLOW_OPTIMIZE_ENTRY_BLOCK [V2.06.00 or later]

-OPtimize / -NOOPtimize [V2.01.00 or later]

This option specifies whether link-time optimization is to be executed.

[Specification format]

```
-OPtimize[=SYmbol_delete]
-NOOPtimize
```

- Interpretation when omitted
  Link-time optimization is executed. It is the same result as when the -optimize option is specified.

[Detailed description]

- This option executes link-time optimization (inter-module optimization) for a file for which the -goptimize option was specified in the process of compilation or assembly.
- The -optimize option performs optimization.
  The -nooptimize option suppresses optimization.
- At link-time optimization, CC-RH deletes variables or functions that have not been referenced even once in the program.
- The -entry option used to specify the start address is necessary for searching for variables or functions that have not been referenced.
- *symbol_delete* can be specified as the parameter. However, it has the same meaning as when no parameter is specified.
- If the -optimize or -nooptimize option is specified more than once, the last specification is valid.
- A warning will be output and link-time optimization not performed in the following case.
  - When the -entry option is not specified
- An error will occur in the following case.
  - When an invalid string is specified as the parameter of the -optimize option

-SEction_forbid [V2.01.00 or later]

This option suppresses link-time optimization of specific sections.

[Specification format]

```
-SEction_forbid=sub [, ...]
    sub : = ( section-name[, ...] )
       |   file ( section-name[, ...] )
       |   module ( section-name[, ...] )
```

- Interpretation when omitted
  None

[Detailed description]

- This option suppresses link-time optimization of specific sections.
- Specify a section for which you wish to suppress link-time optimization with *section-name*.
- Specifying *file* or *module* prior to *section-name* allows link-time optimization to be suppressed for sections included in a specific input file or library module.
  - Specify the input file name exactly as it is shown in the link map file, with uppercase and lowercase characters distinguished.
    [Output example of link map file]

    ```
    *** Options ***
    -Input=DefaultBuild\main.obj
    ```
- 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 *section-name*, *file*, or *module* cannot be found
  - When *section-name* is not specified
- A warning will be output and this option ignored in the following case.
  - When the -nooptimize option is specified

[Example of use]

- To suppress link-time optimization of the .SEC1 section, describe as:

  ```
  >rlink a.obj b.obj -optimize -section_forbid=(.SEC1)
  ```
- To suppress link-time optimization of the .SEC1 and .SEC2 sections in a.obj, describe as:

  ```
  >rlink a.obj b.obj -optimize -section_forbid=a.obj(.SEC1,.SEC2)
  ```

-Absolute_forbid [V2.01.00 or later]

This option suppresses link-time optimization in a specific address range.

[Specification format]

```
-Absolute_forbid=range [, ...]
    range := address[+size]
```

- Interpretation when omitted
  None

[Detailed description]

- This option suppresses link-time optimization in a specific address range.
- Specify the range in which you wish to suppress link-time optimization with *address* and *size*. Optimization will be suppressed for the sections included in the range of "*address* + *size*".
- Specify *address* and *size* as hexadecimal values from 0 to ffffffff.
- If +*size* is omitted, it is assumed that +0 was specified.
- If this option is specified more than once, all specifications will be valid.
- A warning will be output and this option ignored in the following case.
  - When the -nooptimize option is specified

[Remark]

- When the range specified by this option overlaps with overlaid sections specified by the -start option, optimization will be suppressed for all overlaid sections in the overlapped area.
  To suppress optimization of only specific sections, use the -section_forbid option.

[Example of use]

- To suppress link-time optimization in the range of addresses 0x1000 to 0x11ff, describe as:

```
>rlink a.obj b.obj -optimize -absolute_forbid=1000+200
```

-SYmbol_forbid [V2.01.00 or later]

This option suppresses link-time optimization of specific symbols.

[Specification format]

```
-SYmbol_forbid=symbol[, ...]
```

- Interpretation when omitted
  None

[Detailed description]

- This option suppresses link-time optimization of specific symbols.
- Specify the variable names or function names you wish not to delete in link-time optimization with *symbol*. For a variable name or function name defined in the C language, add prefix "_" to the definition name in the program.
- Variables or functions that are referenced by the variables or functions that were specified by *symbol* are also not deleted.
- 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 *symbol* cannot be found
  - When *symbol* is not specified
- A warning will be output and this option ignored in the following case.
  - When the -nooptimize option is specified

[Example of use]

- To suppress link-time optimization of the C-language function "sub()", describe as:

```
>rlink a.obj b.obj -optimize -symbol_forbid=_sub
```

-ALLOW_OPTIMIZE_ENTRY_BLOCK [V2.06.00 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.

[Remarks]

- 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.

[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
```

Section specification

The section specification options are as follows.

- -STARt
- -FSymbol
- -ALIGNED_SECTION

-STARt

This option specifies the start address of the section.

[Specification format]

```
-STARt=suboption [, ...]
  suboption := placement-unit[, ...][/address]
  placement-unit := overlay-sections
                  |  section
  overlay-sections := ( section-list : 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.

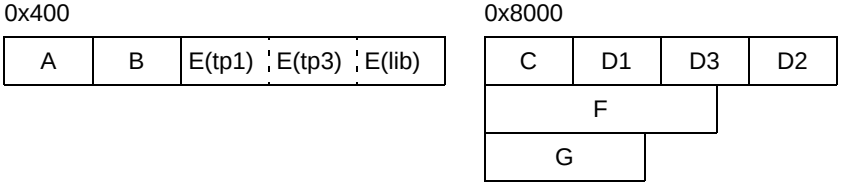
[Detailed description]

- 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 (specifying 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

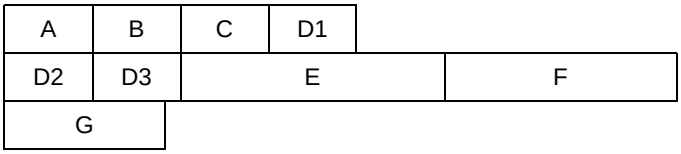


  - Sections C, F, and G delimited by ":" are allocated to the same address.
  - Sections specified with wildcard characters (in this example, the sections whose names start with "D") are allocated in the input order.
  - Objects in the sections having the same name (section E in this example) are allocated 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

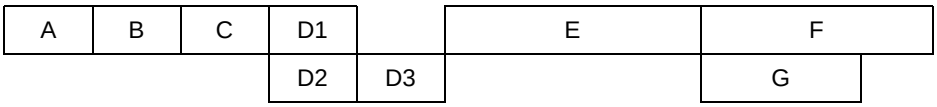
0x400



- 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 this option is specified with parentheses, optimization by the linkage editor 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 "sct2" and "sct3" to symbol address file "test.fsy", describe as:

```
>rlink a.obj b.obj -fsymbol=sct2,stc3 -output=test.abs
```

The output example of symbol address file "test.fsy" is shown below.

```
;RENESAS OPTIMIZING LINKER GENERATED FILE xxxx.xx.xx
;fsymbol = sct2,sct3

;SECTION NAME = sct2
    .public _f
_f .equ 0x0
    .public _g
_g .equ 0x16
;SECTION NAME = sct3
    .public _main
_main .equ 0x20
```

[Remark]

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

-ALIGNED_SECTION

This option changes the number of alignment of the section to 16 bytes.

[Specification format]

```
-ALIGNED_SECTION=section[,section]...
```

- Interpretation when omitted
  None

[Detailed description]

- This option changes the number of alignment of section *section* to 16 bytes.

[Example of use]

- To change the number of alignment of section A to 16 bytes, describe as:

```
>rlink a.obj b.obj -aligned_section=A
```

[Remark]

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

Verify specification

The verify specification option is as follows.

- -CPu

-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
  None

[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]

- If the -form={object|relocate|library} option or -strip option is 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 "test.sub" 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 test.sub, describe as:

```
>rlink file1.obj -subcommand=test.sub 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
```

Other

Other options are as follows.

- -S9
- -STACk
- -COmpress
- -NOCOmpress
- -MEMory
- -REName
- -LIB_REName [V2.01.00 or later]
- -DELete
- -REPlace
- -EXTract
- -STRip
- -CHange_message
- -Hide
- -Total_size
- -VERBOSE [V2.03.00 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 address of the entry point exceeds 0x10000.

[Example of use]

- To output the S9 record at the end even if the address of the entry point exceeds 0x10000, describe as:

```
>rlink a.obj b.obj -form=stype -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 -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 -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 -nocompress
```

-MEMory

This option specifies the memory size occupied during linking.

[Specification format]

```
-MEMory=[occupancy]
```

- Interpretation when omitted
  The processing is the same as usual (It is the same result 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 processing is the same as usual.
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 -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 -optimize, -compress, -delete, -rename, -lib_rename, -map, or -stack options
    - Combination of the -list option and the -show={reference|xreference|struct} 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 2.5 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 external 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
  - When this option is specified together with the -extract or -strip option.

[Example of use]

- To change symbol name "_sym1" to "data", describe as:

```
>rlink a.obj b.obj -rename=(_sym1=data)
```

- To change section ".text" in library module "lib1" to "data" to section "P", describe as:

```
>rlink -library=lib.lib -rename=lib1(.text=P) -form=library
```

[Remark]

- 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 -Xmerge_files.

-LIB_REName [V2.01.00 or later]

This option changes a symbol name or section name that was input from a library.

[Specification format]

```
-lib_rename=name1=name2[, ...]
-lib_rename=file(name1=name2[, ...])
-lib_rename="file|module[|module ...](name1=name2[, ...])"
```

- Interpretation when omitted
  None

[Detailed description]

- This option changes the name of an external 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 the change as *name2*.
- When you specify a C variable name, add prefix "_" to 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
  - When this option is specified together with the -form={object,library}, -extract, or -strip option

[Remark]

- 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 compiler option -Xmerge_files.

[Example of use]

- To change "_sym1" in b.lib and c.lib to "_data", describe as:

```
>rlink a.obj -lib=b.lib,c.lib  -lib_rename=(_sym1=_data)
```

- To change "_sym1" in all modules in b.lib to "_data", describe as:

```
>rlink a.obj -lib=b.lib,c.lib  -lib_rename=b.lib(_sym1=_data)
```

- To change "_sym1" in modules m1 and m2 in b.lib to "_data", describe as:

```
>rlink a.obj -lib=b.lib,c.lib  -lib_rename="b.lib|m1|m2(_sym1=_data)"
```

-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 -Xmerge_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=lib1.lib,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 -Xmerge_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".
- If this option is specified, one input file should correspond to one output file.

[Example of use]

- To delete debug information of file1.abs, file2.abs, and file3.abs, and output these to file1.abs, file2.abs, and file3.abs, respectively, describe as:
  The files before debug information is deleted are backed up in file1.abk, file2.abk, and file3.abk.
  <Command line>

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

<Subcommand file "sub.txt">

```
-input=file1.abs file2.abs file3.abs
-strip
```

[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:
  - 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
    The entry function name is not hidden.

[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 used if the output file format is relocate or library and the input file was compiled or assembled with the -goptimize option specified.
- 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.

-VERBOSE [V2.03.00 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 "test.sub" with the following content.

```
input=a.obj,b.obj               ;(1)
start=P,C,D/100,B/8000          ;(2)
output=a.abs                    ;(3)
end
input=a.abs                     ;(4)
form=stype                      ;(5)
output=a.mot                    ;(6)
```

To specify subcommand file test.sub, describe as:

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

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 "test.sub" with the following content.

```
input=a.obj,b.obj               ;(1)
start=P,C,D/100,B/8000          ;(2)
output=a.abs                    ;(3)
exit
```

To specify subcommand file test.sub, describe as:

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

2.6 Specifying Multiple Options

This section describes the operation when two or more options are specified for the ccrh command at the same time.

2.6.1 Priority

The following options disable other options.

-V/-h	All options will be invalid. At this time, a warning will not be output.
-P	Since execution is terminated at preprocess processing, options related to the following processing will be invalid. At this time, a warning will be output.
-S	Since execution is terminated at compile processing, options related to the following processing will be invalid. At this time, a warning will be output.
-c	Since execution is terminated at assemble processing, options related to the following processing will be invalid. At this time, a warning will be output.
-Xcpu=g3k	The -Xfloat option will be invalid. At this time, a warning will be output.
-lang=c99	The -Xmisra2004 option will be invalid. At this time, a warning will be output.

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, -Odefault, -Osize, -Ospeed

Depending on the order of specified options, the following options will be invalid.

- -O*item*[Note] that is specified before -Onothing, -Odefault, -Osize, or -Ospeed

Note See "-O" in "2.5.1 Compile options" for details about -*Oitem*. Note, however, that -*Omap* and -*Osmap* will not be affected by -*Olevel*.

2.6.2 Incompatible features

If options are specified by the following combinations, an error will occur.

- -Omap, -Xep=fix
- -Osmap, -Xep=fix
- -Omap and -Xsection=data=ep_disp16 or ep_disp23
- -Osmap and -Xsection=data=ep_disp16 or ep_disp23

2.6.3 Dependencies

The behavior of the following options varies depending on what other options are specified.

-Xpreprocess	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 correct.
-Oinline	If this option is specified at the same time with the -Xmerge_files option, inline expansion may be performed between files.

2.6.4 Relationship with #pragma directives

The behavior of the following options varies depending on the relationship with #pragma directives.

- When the -Xep=callee or -Xep option is not specified
  If the following attribute strings are specified at the same time in a #pragma section, an error will occur.
  edata, edata23, tdata, tdata4, tdata5, tdata7, tdata8, ep_auto, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23

3. OUTPUT FILES

This chapter explains the format and other aspects of lists 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	Location counter value, code, line number, and source program under assembly
Section list	Type, size, and name of section
Command line information	Character string of command line of assembler

3.1.2 Assemble list

The location counter value, code, line number, and source program under assembly is output.
The output example of the assemble list is shown below.

```
(1)       (2)                 (3) (4)
OFFSET    CODE                NO  SOURCE STATEMENT

00000000                       1  #CC-RH Compiler RH850 Assembler Source
00000000                       2  #@      CC-RH Version : VX.XX.XXx [DD Mmm YYY]
00000000                       3  #@      Command : main.c -Xcommon=rh850 -S
00000000                       4  #@      compiled at Sun Jan 1 00:00:00 2012
00000000                       5          .cseg text
00000000                       6          ld.w $_data,r12
00000000  440E0000              -- movhi       0x0,gp,r1
00000004  21670100              -- ld.w        0x0[r1],r12
00000008                       7
00000000                       8          .dseg data
00000000  00000000             9  _data:  .dw 0
00000004                      10
```

Number	Description
(1)	Location counter value The location counter value for the beginning of the code generated for the source program of the corresponding line is output.
(2)	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.
(3)	Line number The number of the line is output. This is expressed in a decimal number.

Number	Description
(4)	Source program The source program of the line is output. If instruction expansion is performed for the instruction at that line, the disassembly of the array of machine language instructions generated after the instruction expansion is displayed after "--". Compiler information (lines 1 to 4) is output only when an assembly source file output from the compiler is assembled.

3.1.3 Section list

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

TEXT            8 (00000008) .text
DATA            4 (00000004) .data
```

Number	Description
(1)	Section type The type of the section is output as the relocation attribute.
(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
a.asm -Xcommon=rh850 -Xprn_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
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 the -show=struct is specified, the addresses of the structure and union members are also 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
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 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=test.sub -list -show
```

<Subcommand file "test.sub">

```
input sample.obj
```

```
*** Options ***

-subcommand=test.sub    (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 "test.sub" are output.

3.2.3 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.

3.2.4 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

.text
                                 00000000  0000003b      3c    2
.data
                                 fe600006  fe600003       4    4
.bss
                                 fe600004  fe60000b       8    4
```

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 ***

                                                                 (6)
SECTION                          START     END         SIZE    ALIGN   ATTRIBUTE
.text
                                 00000100  0000013b      3c    2       TEXT
.data
                                 000f0400  000f0403       4    4       DATA
.bss
                                 000f0404  000f040b       8    4       BSS
```

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.
(6)	Relocation attribute type of section The relocation attribute of the section which is categorized into the five types of TEXT, CONST, DATA, BSS and OTHER is output.

3.2.5 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.

3.2.6 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
                00000000    00000023        24
  _main
                00000000          0         func ,g            0
  _func_01
                00000018          0         func ,g            0
SECTION=.bss
FILE=sample.obj
                fe600004    fe60000b        8
  _gvall
                fe600004          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 information regarding the structure and union members defined in the files that have been compiled with -g specified is also output. An output example of the structure member information is shown below.

```
*** Symbol List ***
SECTION=(1)
FILE=(2)                                START     END      SIZE
                                         (3)       (4)      (5)
  SYMBOL                                  ADDR      SIZE     INFO   COUNTS OPT
   (6)                                    (7)       (8)      (9)     (10)  (11)
    STRUCT                                          SIZE
     (12)                                           (13)
      MEMBER                              ADDR      SIZE     INFO
       (14)                               (15)      (16)     (17)
SECTION=.bss
FILE=C:¥Users¥b1501079¥Desktop¥a.obj
                                  00001000 00001007         8
 _st
                                  00001000        8 data ,g      0
   struct {
                                                  8
      _st.mem1
                                  00001000        4 int
      _st.mem2
                                  00001004        2 short
      _st.stmem
                                  00001006        2
   struct {
                                                  2
      _st.stmem.mem3
                                  00001006        1 char
      _st.stmem.mem4
                                  00001007        1 char
}
}
```

Number	Description
(12)	struct is output for a structure or union is output for a union.
(13)	Total size of the structure or union
(14)	The member name is concatenated after a symbol name with a dot (.).
(15)	The member address is output.
(16)	The member size is output.
(17)	The member type is output.

3.2.7 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.

3.2.8 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
                                 00000000
                 _func_01
                                 00000018
     SECTION=.data
                 _gval3
                                 fe600000  0003(00000032:.text)
                                           0003(00000038:.text)
     SECTION=.bss
                 _gval1
                                 fe600004  0001(0000001a:.text)
                                           0001(00000020:.text)
                 _gval2
                                 fe600008  0002(00000026:.text)
                                           0002(0000002c:.text)
0002 sample2
     SECTION=.text
                 _func02
                                 00000024  0001(0000000a:.text)
0003 sample3
     SECTION=.text
                 _func03
                                 00000030  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.

Number	Description
(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*)

3.2.9 CRC information

When the CRC option is specified, the CRC operation result and its output address are output.
The output example of all items is shown below.

```
*** CRC Code ***

CODE: cb0b
(1)
ADDRESS: 00007ffe
(2)
```

Number	Description
(1)	CRC operation result
(2)	Address of CRC operation result output

3.3 Link Map File (When Objects Are Combined)

This section explains the link map file that is output when the object combine function is used in a multi-core project.
The link map file has detailed information regarding object combining.

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 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=test.sub -list
```

<Subcommand file "test.sub">

```
input sample1.mot
input sample2.mot
form stype
output result
```

```
*** Options ***

-subcommand=test.sub    (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 "test.sub" 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=test.sub -list -show
```

<Subcommand file "test.sub">

```
form library
input extmod1
input extmod2
output usrlib.lib
```

```
*** Options ***

-subcommand=test.sub    (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 "test.sub" 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=RH850               (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

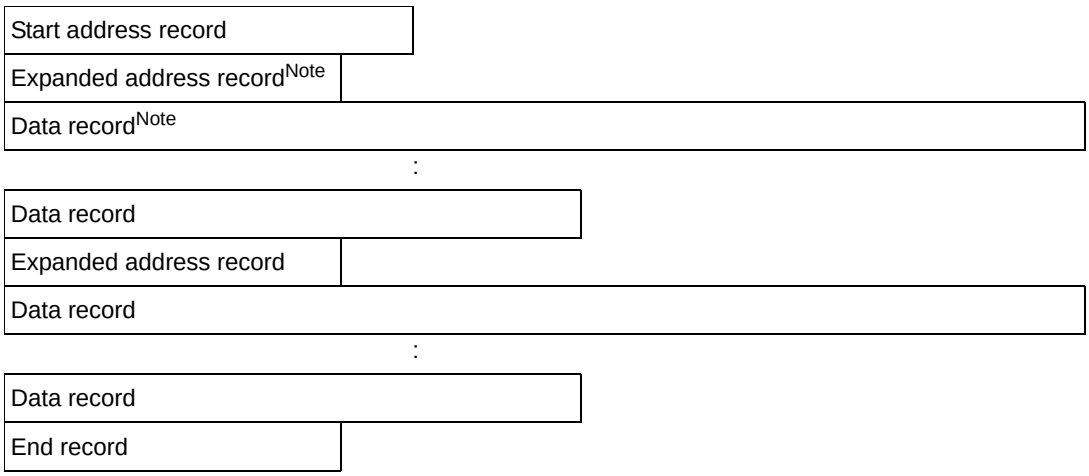
The Intel HEX file (20 bits) consists of four records[Note]: start address record, expanded address record, data record, and end record.

The Intel HEX file (32 bits) consists of six records[Note]: start linear address record, expanded linear start address record, start address record, expanded address record, data record, and end 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 expanded address record and data record are repeated.

Output Information	Description
Start linear address record	Linear address
Expanded linear address record	Upper 16-bit address at bits 32 to 16
Start address record	Entry point address
Expanded address record	Paragraph value of load address
Data record	Value of code
End 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: Expanded linear address record 03: Start address record 02: Expanded address record 00: Data record 01: End 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 expanded address so that a space of up to 1 Mbyte (20 bits) can be used.
Specifically, the record type that specifies the 16-bit expanded address is added.
This expanded 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 Expanded 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 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 Expanded 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 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

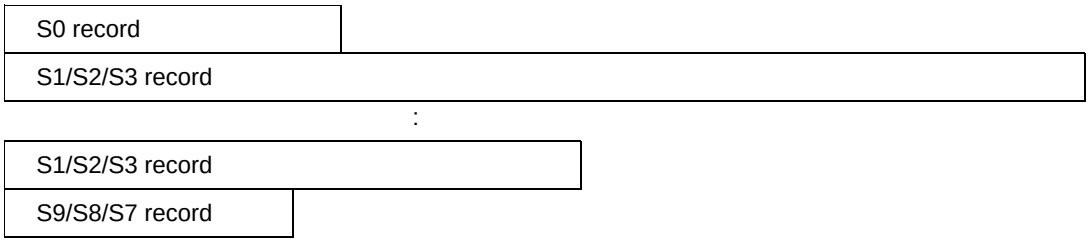
The Motorola S-record file consists of seven records[Note 1]: S0 record as the header record, S1, S2, and S3 records as the data record, and S9, S8, and S7 records as the end records[Note 2].

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] is expressed.
(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.

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

4. COMPILER LANGUAGE SPECIFICATIONS

This chapter explains Compiler language specifications (basic language specification, extended language specifications, etc.) supported by the CC-RH.

4.1 Basic Language Specifications

This section explains the implementation-defined behavior of the CC-RH which is compliant with the C90 and C99 standards.
See "4.2 Extended Language Specifications" for extended language specifications explicitly added by the CC-RH.

4.1.1 Implementation-defined behavior of C90

(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.

Expanded Notation	Value (ASCII)
\a	0x07
\b	0x08
\f	0x0C
\n	0x0A
\r	0x0D
\t	0x09
\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 integer character constant consisting of up to four characters has a four-byte value with the lower byte being the last character and the upper byte being the first character. A character constant having five or more characters results in an error. A character which is not represented by basic execution environment character set is regarded as a integer character constant having that value. In an invalid escape sequence, the backslash is ignored and the next character is regarded as a integer 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 signed char type.

(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).

As per the option (-Xround) specification and microcomputer settings.

(22) The direction of truncation or rounding when a floating-point number is converted to a narrower floating-point number (6.2.1.4).

As per the option (-Xround) specification and microcomputer settings.

(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 long type.

(24) The result of casting a pointer to an integer or vice versa (6.3.4).

Integer-to-pointer conversion result
If the size of an integer type is larger than that of a pointer type, the lower-byte value of the integer type is used. If the size of the integer type is equal to that of the pointer type, the bit pattern of the integer type is retained as is. If the size of the integer type is smaller than that of the pointer type, the resultant value of an extension to an int type is retained as is.

Pointer-to-integer conversion result
If the size of a pointer type is larger than that of an integer type, the lower-byte value of the pointer type is used. If the size of the pointer type is equal to that of the integer type, the bit pattern of the pointer type is retained as is. If the size of a pointer type is smaller than that of an integer type, the zero-extended value of the pointer type is used.

(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).

Optimize for the fastest possible access, regardless of whether there is a register specifier.

(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 a signed int type. The most significant bit of the bit field is treated as a sign bit.

(30) The order of allocation of bit-fields within an int (6.5.2.1).

Allocated from the lower order. Selectable by option -Xbit_order or #pragma bit_order.

(31) Whether a bit-field can straddle a storage-unit boundary (6.5.2.1).

A bit-field cannot straddle a strage-unit boundary, but it is allocated to the next area.

(32) The integer type chosen to represent the values of an enumeration type (6.5.2.2).

signed int type. However, the minimum type that an enumerated type fits in if option -Xenum_type=auto is specified.

(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).

2147483647.

(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 can be a negative value.

(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

(38) The support for quoted names for includable source files (6.8.2).

Searced in this order:
1. Folder specified by the path (if it is full-path)
2. Folder where source file exists
3. Folder specified by -I
4. Standard include file folder

(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 "4.2.6 Using extended language specifications".

(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 "7.4.10 Mathematical functions".

(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 functions.

(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).

Decimal notation.

(65) The input for %p conversion in the fscanf function (7.9.6.2).

Decimal integer.

(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 "sscanf" in "7.4.7 Standard I/O functions".

(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 the description of perror in "7.4.7 Standard I/O functions".

(69) The behavior of the calloc, malloc, or realloc function if the size requested is zero (7.10.3).

The calloc or malloc function returns the allocated pointer, assuming the request size is 8. The realloc function returns NULL.

(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 the description of the strerror function in "7.4.3 Character string functions".

(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.

The table below shows the translation limits of CC-RH.
The upper limit depends on the memory situation of the host environment for the item "No limit".

item	limit
Number of nesting levels of conditional inclusion	No limit
Number of pointers, arrays, and function declarators (in any combinations) qualifying an arithmetic, structure, union, or incomplete type in a declaration	128
Number of nesting levels of parenthesized declarators within a full declarator	No limit
Number of nesting levels of parenthesized expressions within a full expression	No limit
Number of significant initial characters in an internal identifier or a macro name	No limit
Number of significant initial characters in an external identifier	No limit
Number of external identifiers in one translation unit	No limit
Number of identifiers with block scope declared in one block	No limit
Number of macro identifiers simultaneously defined in one preprocessing translation unit	No limit
Number of parameters in one function definition	No limit
Number of arguments in one function call	No limit
Number of parameters in one macro definition	No limit
Number of arguments in one macro invocation	No limit
Number of characters in a logical source line	No limit
Number of characters in a character string literal or wide string literal (after concatenation)	No limit
Number of bytes in an object (in a hosted environment only)	2147483647
Number of nesting levels for #included files	No limit
Number of case labels for a switch statement (excluding those for any nested switch statements)	2147483647
Number of members in a single structure or union	No limit
Number of enumeration constants in a single enumeration	No limit
Number of levels of nested structure or union definitions in a single struct-declaration-list	No limit

4.1.2 Implementation-defined behavior of C99

(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 signed char type.

(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 integer character constant consisting of up to four characters has a four-byte value with the lower byte being the last character and the upper byte being the first character. A character constant having five or more characters results in an error. A character which is not represented by basic execution environment character set is regarded as a integer character constant having that value. In an invalid escape sequence, the backslash is ignored and the next character is regarded as a integer 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).

As per the option (-Xround) specification and microcomputer settings.

(36) The direction of rounding when a floating-point number is converted to a narrower floating-point number (6.3.1.5).

As per the option (-Xround) specification and microcomputer settings.

(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).

As per the option (-Xround) specification.

(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).

Integer-to-pointer conversion result
If the size of an integer type is larger than that of a pointer type, the lower-byte value of the integer type is used. If the size of the integer type is equal to that of the pointer type, the bit pattern of the integer type is retained as is. If the size of the integer type is smaller than that of the pointer type, the resultant value of an extension to an int type is retained as is.

Pointer-to-integer conversion result
If the size of a pointer type is larger than that of an integer type, the lower-byte value of the pointer type is used. If the size of the pointer type is equal to that of the integer type, the bit pattern of the pointer type is retained as is. If the size of a pointer type is smaller than that of an integer type, the zero-extended value of the pointer type is used.

(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 a signed int type. The most significant bit of the bit field is treated as a sign bit.

(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. Selectable by option -Xbit_order or #pragma bit_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).

signed int type. However, the minimum type that an enumerated type fits in if option -Xenum_type=auto is specified.

(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 can be a negative value.

(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 "4.2.6 Using extended language specifications" 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 FUNCTIONAL 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).

The atan2, cos, sin, tan, frexp, pow, lround, llround, fmod functions might result in a domain error.

(72) The values returned by the mathematics functions on domain errors (7.12.1).

For details, refer to "7.4.10 Mathematical functions".

(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. However, the exp or ldexp functions returns 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 functions.

(75) The base-2 logarithm of the modulus used by the remquo functions in reducing the quotient (7.12.10.3).

The remquo functions 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) is output for a positive infinity, (-INF) for a negative infinity, and (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).

Decimal notation.
The fwprintf function is 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 "sscanf" in "7.4.7 Standard I/O functions".
The fwscanf function is 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).

Decimal integer.
The fwscanf function is 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 or strtodf function.
The strtof, strtold, 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 and strtodf functions set ERANGE in global variable errno.
The strtof, strtold, 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).

The calloc or malloc function returns the allocated pointer, assuming the request size is 8. The realloc function returns NULL.

(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).

Depends on the linked library.
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).

The values in parentheses are for the case when the -Xdbl_size=4 option is used, which specifies sizeof(double) = sizeof(long double) = 4.

float.h

Name	Value	Meaning
FLT_ROUNDS	1 -Xround=nearest 0 -Xround=zero	Rounding mode for floating-point addition. 1: Rounded to nearest 0: Rounded to zero
FLT_EVAL_METHOD	0	Evaluation format of floating-point number
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)	
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
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)	

Name	Value	Meaning
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 - 07F)	
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)	

half.h

Name	Value	Meaning
HALF_MANT_DIG	+11	Number of numerals (p) with FLT_RADIX of floating- point mantissa as base
HALF_DIG	+2	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
HALF_MIN_EXP	-13	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.
HALF_MIN_10_EXP	-4	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.
HALF_MAX_EXP	+16	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.
HALF_MAX_10_EXP	+4	Maximum integer that falls in the range of a normalized floating-point number when 10 is raised to this power. $\log_{10} ((1 - b^{-p}) * b^{e_{max}})$
HALF_MAX	65504.0F	Maximum value of finite floating-point numbers that can be expressed $(1 - b^{-p}) * b^{e_{max}}$
HALF_EPSILON	0.00097656F	Difference between 1.0 that can be expressed by specified floating-point number type and the lowest value which is greater than 1. $b^{1 - p}$
HALF_MIN	6.10352E-05F	Minimum value of normalized positive floating-point number $b^{e_{min} - 1}$

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
CHAR_MIN	-128	Minimum value of char
CHAR_MAX	+127	Maximum value of 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	-2147483648	Minimum value of int
INT_MAX	+2147483647	Maximum value of int
UINT_MAX	+4294967295	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
LLONG_MAX	+9223372036854775807	Maximum value of long long int
ULLONG_MAX	+18446744073709551615	Maximum value of unsigned long long int

stdint.h

Name	Value	Meaning
INT8_MIN	-0x7f-1	Minimum value of int8_t
INT16_MIN	-0x7fff-1	Minimum value of int16_t
INT32_MIN	-0x7fffffff-1	Minimum value of int32_t
INT64_MIN	-0x7fffffffffffffffLL-1	Minimum value of int64_t
INT8_MAX	0x7f	Maximum value of int8_t
INT16_MAX	0x7fff	Maximum value of int16_t
INT32_MAX	0x7fffffff	Maximum value of int32_t
INT64_MAX	0x7fffffffffffffffLL	Maximum value of int64_t
UINT8_MAX	0xff	Maximum value of uint8_t
UINT16_MAX	0xffff	Maximum value of uint16_t
UINT32_MAX	0xffffffff	Maximum value of uint32_t
UINT64_MAX	0xffffffffffffffffULL	Maximum value of uint64_t
INT_LEAST8_MIN	-0x7f-1	Minimum value of int_least8_t

Name	Value	Meaning
INT_LEAST16_MIN	-0x7fff-1	Minimum value of int_least16_t
INT_LEAST32_MIN	-0x7fffffff-1	Minimum value of int_least32_t
INT_LEAST64_MIN	-0x7fffffffffffffffLL-1	Minimum value of int_least64_t
INT_LEAST8_MAX	0x7f	Maximum value of int_least8_t
INT_LEAST16_MAX	0x7fff	Maximum value of int_least16_t
INT_LEAST32_MAX	0x7fffffff	Maximum value of int_least32_t
INT_LEAST64_MAX	0x7fffffffffffffffLL	Maximum value of int_least64_t
UINT_LEAST8_MAX	0xff	Maximum value of uint_least8_t
UINT_LEAST16_MAX	0xffff	Maximum value of uint_least16_t
UINT_LEAST32_MAX	0xffffffffU	Maximum value of uint_least32_t
UINT_LEAST64_MAX	0xffffffffffffffffULL	Maximum value of uint_least64_t
INT_FAST8_MIN	-0x7fffffff-1	Minimum value of int_fast8_t
INT_FAST16_MIN	-0x7fffffff-1	Minimum value of int_fast16_t
INT_FAST32_MIN	-0x7fffffff-1	Minimum value of int_fast32_t
INT_FAST64_MIN	-0x7fffffffffffffffLL-1	Minimum value of int_fast64_t
INT_FAST8_MAX	0x7fffffff	Maximum value of int_fast8_t
INT_FAST16_MAX	0x7fffffff	Maximum value of int_fast16_t
INT_FAST32_MAX	0x7fffffff	Maximum value of int_fast32_t
INT_FAST64_MAX	0x7fffffffffffffffLL	Maximum value of int_fast64_t
UINT_FAST8_MAX	0xffffffffU	Maximum value of uint_fast8_t
UINT_FAST16_MAX	0xffffffffU	Maximum value of uint_fast16_t
UINT_FAST32_MAX	0xffffffffU	Maximum value of uint_fast32_t
UINT_FAST64_MAX	0xffffffffffffffffULL	Maximum value of uint_fast64_t
INTPTR_MIN	-0x7fffffff-1	Minimum value of intptr_t
INTPTR_MAX	0x7fffffff	Maximum value of intptr_t
UINTPTR_MAX	0xffffffffU	Maximum value of uintptr_t
INTMAX_MIN	-0x7fffffffffffffffLL-1	Minimum value of intmax_t
INTMAX_MAX	0x7fffffffffffffffLL	Maximum value of intmax_t
UINTMAX_MAX	0xffffffffffffffffULL	Maximum value of uintmax_t
PTRDIFF_MIN	-0x7fffffff-1	Minimum value of ptrdiff_t
PTRDIFF_MAX	0x7fffffff	Maximum value of ptrdiff_t
SIZE_MAX	0xffffffffU	Maximum value of size_t

(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-RH.
The upper limit depends on the memory situation of the host environment for the item "No limit".

item	limit
Number of nesting levels of blocks	No limit
Number of nesting levels of conditional inclusion	No limit
Number of pointers, arrays, and function declarators (in any combinations) qualifying an arithmetic, structure, union, or incomplete type in a declaration	128
Number of nesting levels of parenthesized declarators within a full declarator	No limit
Number of nesting levels of parenthesized expressions within a full expression	No limit
Number of significant initial characters in an internal identifier or a macro name	No limit
Number of significant initial characters in an external identifier	No limit
Number of external identifiers in one translation unit	No limit
Number of identifiers with block scope declared in one block	No limit
Number of macro identifiers simultaneously defined in one preprocessing translation unit	No limit
Number of parameters in one function definition	No limit
Number of arguments in one function call	No limit
Number of parameters in one macro definition	No limit
Number of arguments in one macro invocation	No limit
Number of characters in a logical source line	No limit
Number of characters in a character string literal or wide string literal (after concatenation)	No limit
Number of bytes in an object (in a hosted environment only)	2147483647
Number of nesting levels for #included files	No limit
Number of case labels for a switch statement (excluding those for any nested switch statements)	2147483647
Number of members in a single structure or union	No limit
Number of enumeration constants in a single enumeration	No limit
Number of levels of nested structure or union definitions in a single struct-declaration-list	No limit

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-RH.

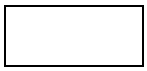
(1) Integer type

(a) Internal representation

The leftmost bit in an area is a sign bit with a signed type (type declared without "unsigned"). The value of a signed type is expressed as 2' s complement.

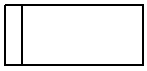
Internal Representation of Integer Type are shown below.

- _Bool

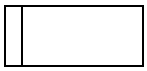


Only the 0th bit has meaning. Bits 1 to 7 are undefined.
If the -lang=c option is specified simultaneously with the -strict_std option, _Bool type will cause a C90 violation error.

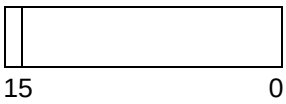
- char



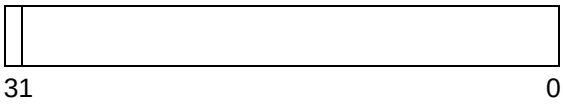
- signed char (no sign bit for unsigned)



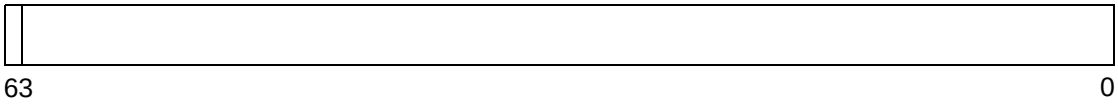
- short (no sign bit for unsigned)



- int, long (no sign bit for unsigned)



- long long (no sign bit for unsigned)



If the -lang=c option is specified simultaneously with the -strict_std option, long long type will cause a C90 violation error.

(b) Value area

Table 4.1 Value Area of Integer Type

Type	Value Area
char[Note]	-128 to +127
short	-32768 to +32767
int	-2147483648 to +2147483647
long	-2147483648 to +2147483647
long long	-9223372036854775808 to +9223372036854775807
unsigned char	0 to 255
unsigned short	0 to 65535
unsigned int	0 to 4294967295
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 4.2 Types of Integer Constants (when -lang=c is specified and -strict_std is not specified)

Suffix	Decimal 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

Table 4.3 Types of Integer Constants (when -lang=c and -strict_std are specified)

Suffix	Decimal 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 4.4 Types of Integer Constants (when -lang=c99 is specified)

Suffix	Decimal 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 IEEE754[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.

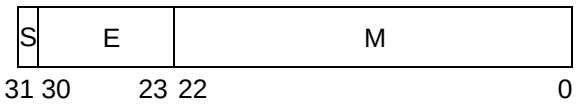
Note IEEE: Institute of Electrical and Electronics Engineers
IEEE754 is a standard to unify specifications such as the data format and numeric range in systems that handle floating-point operations.

Internal Representation of Floating-Point Type are shown below.

- __fp16

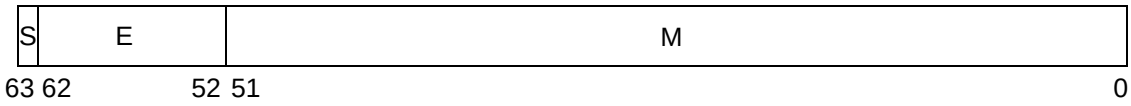
S: Sign bit of mantissa
E: Exponent (5 bits)
M: Mantissa (10 bits)

- 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)

(b) Value area

Table 4.5 Value Area of Floating-Point Type

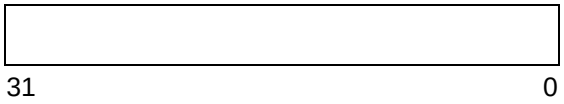
Type	Value Area
__fp16	6.10352e-05F to 65504.0
float	1.17549435E-38F to 3.40282347E+38F
double	2.2250738585072014E-308 to 1.7976931348623158E+308
long double	2.2250738585072014E-308 to 1.7976931348623158E+308

When -Xdbl_size=4 is specified, the double and long double types have the same internal representation and the same value area as those of the float type.

(3) Pointer type

(a) Internal representation
The internal representation of a pointer type is the same as that of an unsigned int type.

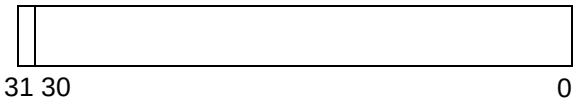
Figure 4.1 Internal Representation of Pointer Type



(4) Enumerate type

(a) Internal representation
The internal representation of an enumerate type is the same as that of a signed int type. The leftmost bit in an area of a sign bit.

Figure 4.2 Internal Representation of Enumerate Type



When the -Xenum_type=auto option is specified, see "(32) The integer type chosen to represent the values of an enumeration type (6.5.2.2).".

(5) Array type

(a) Internal representation

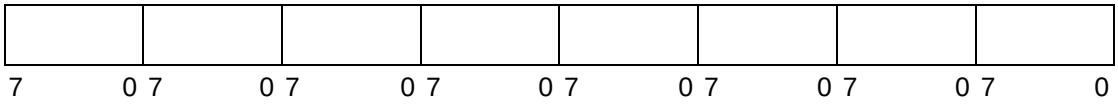
The internal representation of an array type arranges the elements of an array in the form that satisfies the alignment condition (alignment) of the elements

Example

```
char    a[8] = {1, 2, 3, 4, 5, 6, 7, 8};
```

The internal representation of the array shown above is as follows.

Figure 4.3 Internal Representation of Array Type



(6) Structure type

(a) Internal representation

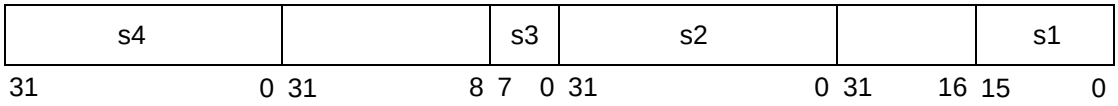
The internal representation of a structure type arranges the elements of a structure in a form that satisfies the alignment condition of the elements.

Example

```
struct {
        short   s1;
        int     s2;
        char    s3;
        long    s4;
} tag;
```

The internal representation of the structure shown above is as follows.

Figure 4.4 Internal Representation of Structure Type



For the internal representation when the structure type packing function is used, see "4.2.6.8 Structure type packing".

(7) Union type

(a) Internal representation

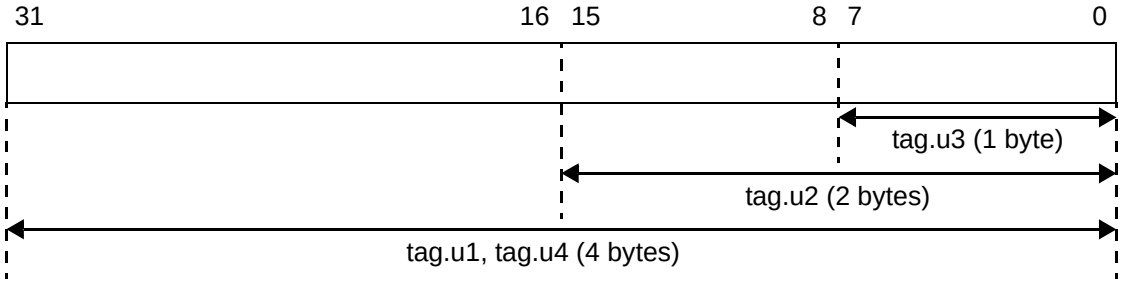
A union is considered as a structure whose members all start with offset 0 and that has sufficient size to accommodate any of its members. The internal representation of a union type is like each element of the union is placed separately at the same address.

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 4.5 Internal Representation of Union Type



(8) Bit field

(a) Internal representation

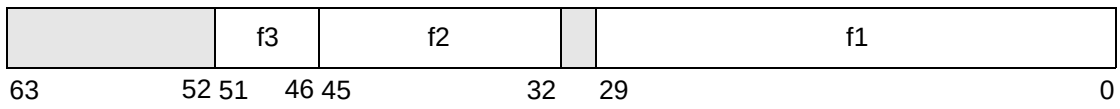
The most significant bit of a bit field declared as a signed type, or without an explicit sign declaration, will be the sign bit. The first bit field to be declared will be allocated starting from the least significant bit in the area with the sign of the type when the bit field was declared. If the alignment condition of the type specified in the declaration of a bit field is exceeded as a result of allocating an area that immediately follows the area of the preceding bit field to the bit field, the area is allocated starting from a boundary that satisfies the alignment condition.
You can allocate the members of a bit field starting from the most significant bit using the -Xbit_order=left option or by specifying #pragma bit_order left. See "4.2.6.9 Bit field assignment" for details.

Example 1.

```
struct {
        unsigned int    f1:30;
        int             f2:14;
        unsigned int    f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.6 Internal Representation of Bit Field

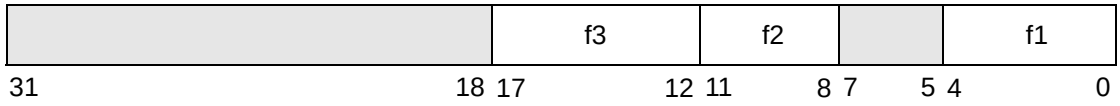


Example 2.

```
struct {
    int             f1:5;
    char            f2:4;
    int             f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.7 Internal Representation of Bit Field



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. However, only signed int and unsigned int types can be specified when the -lang=c option and -strict_std option are specified.
For the internal representation of bit field when the structure type packing function is used, see "4.2.6.8 Structure type packing".

(9) Alignment condition

(a) Alignment condition for basic type
Alignment condition for basic type is as follows.
If the -Xinline_strcpy option of the CC-RH is specified, however, all the arrey types are 4-byte boundaries.

Table 4.6 Alignment Condition for Basic Type

Basic Type	Alignment Conditions
(unsigned) char and its array type _Bool type	Byte boundary
(unsigned) short and its array type	2-byte boundary
Other basic types (including pointer) (unsigned) long long and its array type double and its array type	4-byte boundary
long double and its array type	4-byte boundary

(b) 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.

Here are examples of the respective cases:

Example 1.

```
union  tug1 {
  unsigned short i; /*2 bytes member*/
  unsigned char  c; /*1 bytes member*/
};  /*The union is aligned with 2-byte.*/
```

Example 2.

```
union  tug2 {
  unsigned int  i;  /*4 bytes member*/
  unsigned char  c; /*1 byte member*/
};  /*The union is aligned with 4-byte.*/
```

(c) 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.

Here are examples of the respective cases:

Example 1.

```
struct  ST {
        char    c;      /*1 byte member*/
};  /*Structure is aligned with 1-byte.*/
```

Example 2.

```
struct  ST {
        char    c;      /*1 byte member*/
        short   s;      /*2 bytes member*/
};  /*Structure is aligned with 2-byte.*/
```

Example 3.

```
struct  ST {
        char    c;      /*1 byte member*/
        short   s;      /*2 bytes member*/
        short   s2;     /*2 bytes member*/
};  /*Structure is aligned with 2-byte.*/
```

Example 4.

```
struct  ST {
        char    c;      /*1 byte member*/
        short   s;      /*2 bytes member*/
        int     i;      /*4 bytes member*/
};  /*Structure is aligned with 4-byte.*/
```

Example 5.

```
struct  ST {
        char        c;      /*1 byte member*/
        short       s;      /*2 bytes member*/
        int         i;      /*4 bytes member*/
        long long   ll;     /*4 bytes member*/
};  /*Structure is aligned with 4-byte.*/
```

(d) Alignment condition for function argument
The alignment condition for a function argument is a 4-byte boundary.

(e) Alignment condition for executable program
The alignment condition when an executable object module file is created by linking object files is 2-byte boundary.

4.1.4 Register mode

The CC-RH provides three register modes. By specifying these register modes efficiently, the contents of some registers do not need to be saved or restored when an interrupt occurs or the task is switched. As a result, the processing speed can be improved. The register modes are specified by using the register mode specification option (-Xreg_mode) of the CC-RH. This function reduces the number of registers internally used by the CC-RH on a step-by-step basis. As a result, the following effects can be expected:

- The registers not used can be used for the application program (that is, a source program in assembly language).
- The overhead required for saving and restoring registers can be reduced.

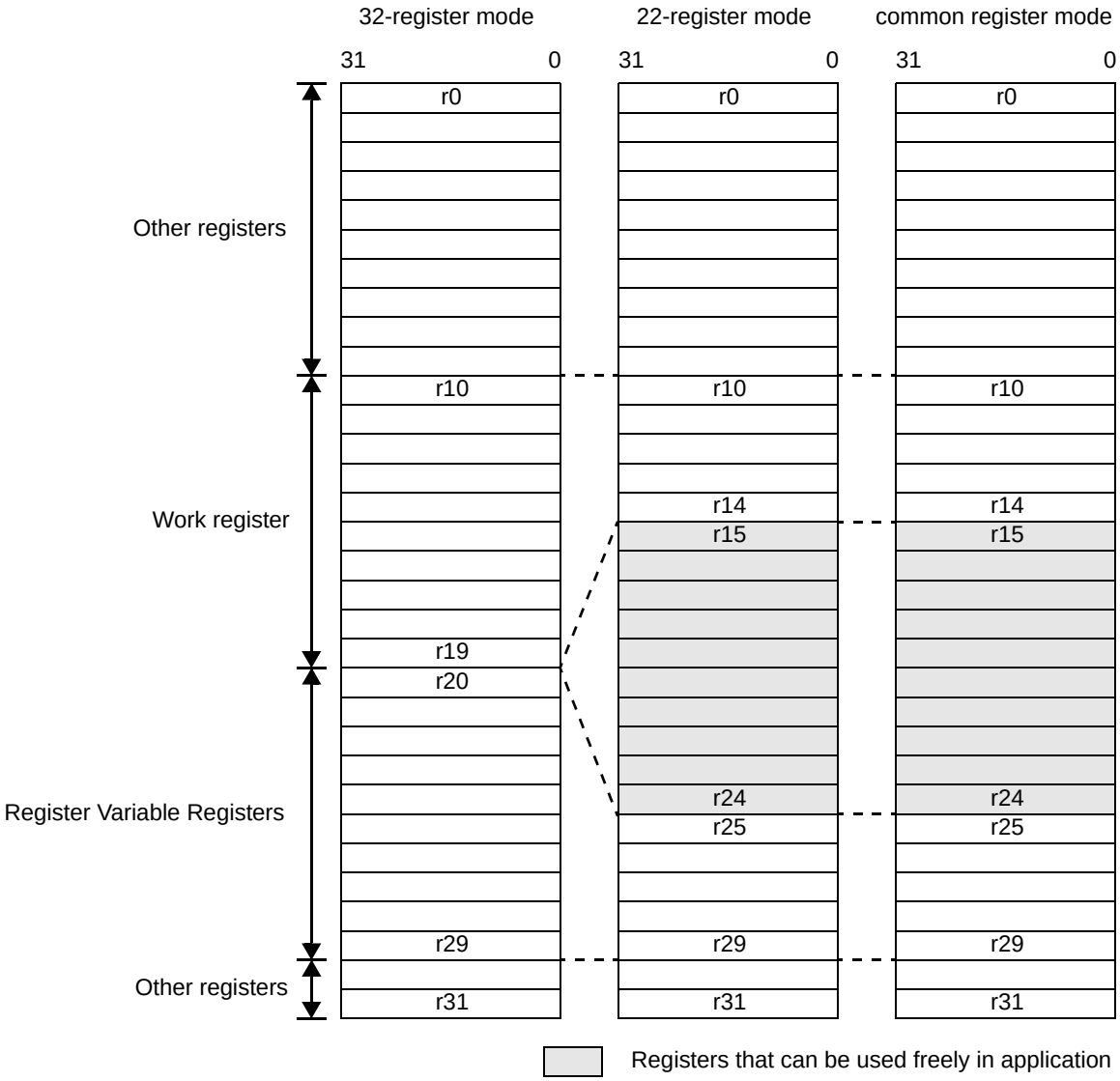
Caution In an application program that has many variables to be allocated to registers by the CC-RH, the variables so far allocated to a register are accessed from memory when a register mode has been specified. As a result, the processing speed may drop.

Next table and next Figure show the three register modes supplied by the CC-RH.

Table 4.7 Register Modes Supplied by CC-RH

Register Modes	Work Registers	Register Variable Registers
32-register mode (Default)	r10 to r19	r20 to r29
22-register mode	r10 to r14	r25 to r29
common register mode	r10 to r14	r25 to r29

Figure 4.8 Register Modes and Usable Registers



Specification example on command line

```
> ccrh -Xreg_mode=22 file.c <- compiled in 22-register mode
```

4.2 Extended Language Specifications

This section explains the extended language specifications supported by the CC-RH.

4.2.1 Reserved words

In CC-RH, all names that start with an underscore (_) and an uppercase alphabetic character, and all names that contain two underscores (__) are invalid as label, variable, or function identifiers.

4.2.2 Macro

The predefined macro names in CC-RH are listed below.

Table 4.8 List of Supported Macros

Condition to be Defined[Note 1]	Macro Name	Value
At all times	__LINE__	Line number of source line at that point (decimal)
At all times	__FILE__	Name of source file (character string constant)
At all times	__DATE__[Note 2]	Date of translating source file (character string constant)[Note 3]
At all times	__TIME__[Note 2]	Translation time of source file (character string constant)[Note 4]
When -strict_std is specified	__STDC__	1
When -lang=c99 is specified	__STDC_HOSTED__	0
At all times	__STDC_VERSION__	199409L (when -lang=c99 is not specified) 199901L (when -lang=c99 is specified)
When -lang=c99 is specified	__STDC_IEC_559__	1
At all times	__RENESAS__	1
At all times	__RENESAS_VERSION__	0xXXYYZZ00[Note 5]
At all times	__CCRH__	1
	__CCRH	1
At all times	__RH850__	1
	__RH850	1
At all times	__v850e3v5__	1
	__v850e3v5	1
When -Xdbl_size=4 is specified	__DBL4	1
	__DOUBLE_IS_32BITS__	1
When -Xdbl_size=8 is specified	__DBL8	1
	__DOUBLE_IS_64BITS__	1
When -Xround=nearest is specified	__RON	1
When -Xround=zero is specified	__ROZ	1
When -Xbit_order=left is specified	__BITLEFT	1

Condition to be Defined[Note 1]	Macro Name	Value
When -Xbit_order=right is specified	__BITRIGHT	1
When -Xenum_type=auto is specified	__AUTO_ENUM	1
When -Xfloat=fpu is specified	__FPU	1
At all times	__CHAR_SIGNED__	1
When -Xreg_mode=32 is specified	__reg32__	1
When -Xreg_mode=22 is specified	__reg22__	1
When -Xreg_mode=common is specified	__reg_common__	1
At all times	_LIT	1
At all times	__MULTI_LEVEL__	Value specified by *level* (decimal) (The value is 0 when the -Xmulti_level option is not specified.)
When -pic is specified	__PIC	1
When -pirod is specified	__PIROD	1
When -pid is specified	__PID	1

Note 1. The condition specified by the option is applicable also for the interpretation when the option is omitted.

Note 2. The date and time of translation can be obtained in any case; the __DATE__ and __TIME__ macro values are always defined.

Note 3. Character string constant in the form of "Mmm dd yyyy". Here, the name of the month (Mmm) is the same as that generated by the asctime function stipulated by the C language standard (three alphabetic characters with only the first character being a capital letter). The first character of dd is blank if its value is less than 10.

Note 4. Character string constant having a format of "hh:mm:ss" similar to the time created by the asctime function.

Note 5. If the version is V.XX.YY.ZZ, this will be 0xXXYYZZ00.
Example V1.02.03 -> __RENESAS_VERSION__ = 0x01020300

4.2.3 C99 language specifications supported in conjunction with C90

CC-RH 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 character strings
The result of concatenating a character string constant with a wide character string constant is a wide character string constant.

(3) _Bool type
_Bool type is supported.

Note When the -lang=c99 option is specified, data is treated as the strict _Bool type. When the -lang=c option is specified, data is treated as the signed char type in some expressions.

(4) long long int type
long long int type is supported. long long int type is 8-byte of integer type.
Appending "LL" to a constant value is also supported. It is also possible to specify this for bit field types.

(5) Integer promotion
In accordance with support for types _Bool and long long, integer promotion is also in accordance with the C99 specification.

(6) Existing argument expansion
In accordance with support for types _Bool and long long, existing argument expansion is also in accordance with the C99 specification..

- Functions are called after expanding type _Bool_ to type int (4 bytes).
- Functions are called with type (unsigned) long long remaining as an 8 bytes value.

(7) Comma permission behind the last enumeration child 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,};
```

(8) Inline keyword (inline function)
Inline keyword is supported.
This can also be specified using a pragma directive, via the following format.

```
#pragma inline ( function-name [, function-name]... )Note
```

Note The outer parentheses can be omitted.

For the details of expansion specification, see "4.2.6.3 Inline expansion".
When -lang=c99 option is specified, inline keyword is treated as C99 keyword. Otherwise, inline keyword has same meaning as #pragma inline directive.

(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".

4.2.4 Compiler generated symbols

The following is a list of symbols generated by the compiler for use in internal processing.
Symbols with the same names as the symbols below cannot be used.

Table 4.9 Compiler Generated Symbols

Target	Generated Symbol
extern function name	*_function name*
static *function name*	*_function name.num*Note
extern variable name	*_variable name*
static variable in the file	*_variable name.num*Note
static variable in the function	*_variable name.num*Note*.function label*
Label in the function	.BB.LABEL.*num1_num2*Note
String literal or initial values of auto variables	.STR.*num*Note
Compound literal	_ _ _T *num1.num2*Note

Target	Generated Symbol
switch table	.SWITCH.LABEL.*num1_num2*[Note].SWITCH.LABEL.*num1_num2*[Note].END

Note *num*, *num1*, and *num2* are arbitrary numbers.

4.2.5 #pragma directive

Below are #pragma directives supported as extended language specifications. These extended specifications can also be used with the _Pragma operator in C99.

Table 4.10 List of Supported #pragma Directive

#pragma directive	Definition
#pragma section	See "4.2.6.1 Allocation of function and data to section".
#pragma inline_asm	See "4.2.6.2 Describing assembler instruction".
#pragma inline #pragma noinline	See "4.2.6.3 Inline expansion".
#pragma interrupt	See "4.2.4.5 (3) Describing interrupt/exception handler".
#pragma block_interrupt	See "4.2.4.6 (2) Disabling interrupts in entire function".
#pragma pack	See "4.2.6.8 Structure type packing".
#pragma bit_order	See "4.2.6.9 Bit field assignment".
#pragma pmodule	See "4.2.6.10 Core number specification (for a multi-core device)".
#pragma align4	See "4.2.6.11 Specifying alignment value for branch destination addresses".
#pragma stack_protector #pragma no_stack_protector	See "4.2.6.12 Detection of stack smashing [Professional Edition only]".
#pragma register_group	See "4.2.6.14 Detection of writing to control registers or insertion of synchronization processing [Professional Edition only] [V1.06.00 or later]".

4.2.6 Using extended language specifications

This section explains using expanded specifications.

- Allocation of function and data to section
- Describing assembler instruction
- Inline expansion
- Controlling interrupt level
- Interrupt/Exception processing handler
- Disabling or enabling maskable interrupts
- Intrinsic functions
- Structure type packing
- Bit field assignment
- Core number specification (for a multi-core device)
- Specifying alignment value for branch destination addresses
- Detection of stack smashing [Professional Edition only]
- Half-precision floating-point type [Professional Edition only] [V1.05.00 or later]
- Detection of writing to control registers or insertion of synchronization processing [Professional Edition only] [V1.06.00 or later]

4.2.6.1 Allocation of function and data to section

The CC-RH controls allocation to memory or the access method by allocating functions or data to sections.

- Sections for functions

Table 4.11 Types of Sections for Functions

Section Relocation Attribute	Default Section Name	Access Method	Alignment Condition
text	.text	32-bit relative mode from r0	2
pctext	.pctext	32-bit relative mode from PC	

- The .text section is used by default.
  This section cannot be used by the PIC facility.
- The .pctext section is a section to be used by the PIC facility.

- R0-relative sections for data

Table 4.12 Types of R0-Relative Sections for Data

Section Relocation Attribute	Default Section Name	Data to be Allocated	Access Method	Alignment Condition
zdata	.zdata	Variables with initial value	16-bit relative mode from r0	4
zbss	.zbss	Variables with no initial value		
zdata23	.zdata23	Variables with initial value	23-bit relative mode from r0	
zbss23	.zbss23	Variables with no initial value		
data	.data	Variables with initial value	32-bit relative mode from r0	
bss	.bss	Variables with no initial value		

- The .data and .bss sections are used by default.
- The .zdata, .zbss, .zdata23, and .zbss23 sections are used for access in a short instruction length.
- Any of the R0-relative sections cannot be used by the PID facility.

- EP-relative sections for data

Table 4.13 Types of EP-Relative Sections for Data

Section Relocation Attribute	Default Section Name	Data to be Allocated	Access Method	Alignment Condition
tdata4	.tdata4	Variables with initial value	4-bit relative mode from r30 (EP) The sld or sst instruction can be used.	4
tbss4	.tbss4	Variables with no initial value		
tdata5	.tdata5	Variables with initial value	5-bit relative mode from r30 (EP) The sld or sst instruction can be used.	
tbss5	.tbss5	Variables with no initial value		
tdata7	.tdata7	Variables with initial value	7-bit relative mode from r30 (EP) The sld or sst instruction can be used.	
tbss7	.tbss7	Variables with no initial value		
tdata8	.tdata8	Variables with initial value	8-bit relative mode from r30 (EP) The sld or sst instruction can be used.	
tbss8	.tbss8	Variables with no initial value		
edata	.edata	Variables with initial value	16-bit relative mode from r30 (EP)	
ebss	.ebss	Variables with no initial value		
edata23	.edata23	Variables with initial value	23-bit relative mode from r30 (EP)	
ebss23	.ebss23	Variables with no initial value		
edata32	.edata32	Variables with initial value	32-bit relative mode from r30 (EP)	
ebss32	.ebss32	Variables with no initial value		

- The .tdata* and .tbss* sections are used for access in a short instruction length.
  The sld or sst instruction which is even shorter can be used.
  These sections can be used even by the PID facility.
- The .edata, .ebss, .edata23, and .ebss23 sections are used for access in a short instruction length.
  These sections can be used even by the PID facility.
- The .edata32 and .ebss32 sections are sections to be used by the PID facility.

- GP-relative sections for data

Table 4.14 Types of GP-Relative Sections for Data

Section Relocation Attribute	Default Section Name	Data to be Allocated	Access Method	Alignment Condition
sdata	.sdata	Variables with initial value	16-bit relative mode from r4 (GP)	4
sbss	.sbss	Variables with no initial value		
sdata23	.sdata23	Variables with initial value	23-bit relative mode from r4 (GP)	
sbss23	.sbss23	Variables with no initial value		
sdata32	.sdata32	Variables with initial value	32-bit relative mode from r4 (GP)	
sbss32	.sbss32	Variables with no initial value		

  - The .sdata, .sbss, .sdata23, and .sbss23 sections are used for access in a short instruction length. These sections can be used even by the PID facility.
  - The .sdata32 and .sbss32 sections are sections to be used by the PID facility.

- Sections for constant data

Table 4.15 Types of Sections for Constant Data

Section Relocation Attribute	Default Section Name	Data to be Allocated	Access Method	Alignment Condition
zconst	.zconst	const variables String literal Initial values of auto variables[Note 1]	16-bit relative mode from r0	4
zconst23	.zconst23		23-bit relative mode from r0	
const	.const		32-bit relative mode from r0	
pcconst16	.pcconst16		16-bit relative mode from the __pc_data symbol	
pcconst23	.pcconst23		23-bit relative mode from the __pc_data symbol	
pcconst32	.pcconst32		32-bit relative mode from the __pc_data symbol	

Note 1. The initial values of auto variables are in the .const section when the PIROD facility is not used and are in the .pcconst32 section when the PIROD facility is used. In either case, they cannot be changed.

  - The .const section is used by default. This section cannot be used by the PIROD facility.
  - The .zconst and .zconst23 sections are used for access in a short instruction length. These sections cannot be used by the PIROD facility.
  - The .pcconst16, .pcconst23, and .pconst32 sections are sections to be used by the PIROD facility.

The sections to which the CC-RH allocates functions or data are the four sections .text, .data, .bss, and .const, by default.

The longer the relative distance at access is, the access is in a long instruction length and the code size is increased.

The CC-RH can allocate functions and data to desired sections using the extended facilities. This makes it possible to generate code using the relocation attribute characteristics of each section.

(1) #pragma section directive

Describe the #pragma section directive in the following format.

```
#pragma section data attribute-strings-for-data "user-defined-name" [V2.03.00 or later]
or
#pragma section data attribute-strings-for-data [V2.03.00 or later]
```

- The allocation section of data is changed. The functions and constant data are not affected.

```
#pragma section const attribute-strings-for-constant-data "user-defined-name"
[V2.03.00 or later]
or
#pragma section const attribute-strings-for-constant-data [V2.03.00 or later]
```

- The allocation section of constant data is changed. The functions and constant data are not affected.

- To change the allocation section of functions, describe as:

```
#pragma section attribute-strings "user-defined-name"
or
#pragma section attribute-strings
```

- The allocation sections of functions, data or constant data are changed.
- When specifying the attribute strings for data, the allocation section of constant data is changed to the default section (.const).
- When specifying the attribute strings for constant data, the allocation section of data is changed to the default section (.data or .bss).

```
#pragma section user-defined-name
```

- The relocation attributes of functions, data and constant data are changed to their defaults, and the sections names are changed according to the user defined names.

```
#pragma section text default [V2.03.00 or later]
```

- The allocation section of functions is changed to the default section (.text). The data and constant data are not affected.

```
#pragma section data default [V2.03.00 or later]
```

- The allocation section of data is changed to the default section (.data or .bss). The functions and constant data are not affected.

```
#pragma section const default [V2.03.00 or later]
```

- The allocation section of constant data is changed to the default section (.const). The functions and data are not affected.

```
#pragma section default
or
#pragma section
```

- The allocation sections of functions, data and constant data are changed to the default sections (.text, .data, .bss, .const).

Section relocation attributes are specified using "attribute strings", and section names are specified using "user-defined names". The usable attribute strings vary depending on the specified options.
The relationship between specifiable attribute strings, corresponding section relocation attributes, and options that can be specified as conditions is shown in Table 4.16, Table 4.17, and Table 4.18.
Attribute strings are case-sensitive.

Table 4.16 Relationship between Attribute Strings for Functions and Section Relocation Attributes

Attribute String	Section Relocation Attribute	Specifiable Condition
text	text	When -pic is not specified
pctext [V1.07.00 or later]	pctext	When -pic is specified

Table 4.17 Relationship between Attribute Strings for Data and Section Relocation Attributes

Attribute String	Section Relocation Attribute	Specifiable Condition
r0_disp16	zdata/zbss	When -pid is not specified
r0_disp23	zdata23/zbss23	
r0_disp32	data/bss	
ep_disp4	tdata4/tbss4	When -Xep=fix is specified
ep_disp5	tdata5/tbss5	
ep_disp7	tdata7/tbss7	
ep_disp8	tdata8/tbss8	
ep_disp16	edata/ebss	
ep_disp23	edata23/ebss23	
ep_disp32 [V1.07.00 or later]	edata32/ebss32	When -Xep=fix and -pid are both specified
gp_disp16	sdata/sbss	When -r4=fix is specified
gp_disp23	sdata23/sbss23	
gp_disp32 [V1.07.00 or later]	sdata32/sbss32	When -r4=fix and -pid are both specified

Table 4.18 Relationship between Attribute Strings for Constant Data and Section Relocation Attributes

Attribute String	Section Relocation Attribute	Specifiable Condition
zconst	zconst	When -pirod is not specified
zconst23	zconst23	
const	const	
pcconst16 [V1.07.00 or later]	pcconst16	When -pirod is specified
pcconst23 [V1.07.00 or later]	pcconst23	
pcconst32 [V1.07.00 or later]	pcconst32	

The following characters are usable in user-defined names.

- 0 to 9
- a to z, A to Z
- _
- @
- .

The allocation section of functions or data written after a #pragma section directive is determined in accordance with the following rules.

1. The relocation attribute is determined from the "attribute string".
   (1) For data, *data* or *bss* is automatically selected depending on whether there is an initial value.
   (2) A format with no attribute string has an effect on both functions and data, and the default relocation attribute of the compiler is used.
2. A string representing the relocation attribute is linked after the "user-defined name"[Note].
   (1) If the user-defined name starts with a number from 0 to 9, "_" is added to the beginning.
   (2) If there is no user-defined name, the default section name is used without change.

   Note This is to prevent sections with different section relocation attributes from having the same section name.
3. When the format of #pragma section default or #pragma section is specified, it has an effect on both functions and data, and the default relocation attribute of the compiler and the default section name are used.

A #pragma section directive is valid from the position where it was written up to the position where the next #pragma section directive appears or up to the end of the source file.

Example

```
#pragma section gp_disp16 "foo"
int a = 1;        /* foo.sdata */
int b;            /* foo.sbss */

#pragma section zconst23 "bar"
const int c = 2;  /* bar.zconst23 */

#pragma section text "123"
void func() {}    /* _123.text */

#pragma section baz
int d = 3;        /* baz.data */
int e;            /* baz.bss */
const int f = 4;  /* baz.const */
void func2() {}   /* baz.text */

#pragma section default
int g = 3;        /* .data */
int h;            /* .bss */
const int i = 4;  /* .const */
void func3() {}   /* .text */
```

The effect of a #pragma section directive differs between variables and functions.

- Variable
  If there are multiple declarations and definitions for a variable and a different #pragma section directive is specified for each, the #pragma section directive that appears first is valid.

  Example

```
int x = 1;    /* Variable x is allocated to foo.data */
int y = 2;    /* Variable y is allocated to bar.data */

#pragma section foo
extern int x;

#pragma section bar
extern int x;
extern int y;
```

- Function
  When the -pic option is not specified, #pragma section is valid only for function definitions.
  When the -pic option is specified, similar as to variables, #pragma section is valid for both declarations and definitions.

  [V2.02.00 or later] #pragma section can also be described within a function.
  Therefore, #pragma section within a function is now valid, which was ignored and caused W0520609 to be output in V2.01.00 or earlier.
  Due to this specification change, if there are multiple static variables within the function or multiple string literals, each value can be allocated to separate sections.

  Example

```
void func() {
  #pragma section AAA
  static int aaa;         /* AAA.bss */
  #pragma section BBB
  static int bbb;         /* BBB.bss */
  #pragma section
     ;
}
static int ccc;           /* .bss */
```

4.2.6.2 Describing assembler instruction

With the CC-RH, assembler instruction can be described in the functions of a C source program.

(1) #pragma directives

One of the #pragma directives to embed assembler instructions is #pragma inline_asm.
This treats the function itself as an assembler instruction only, and performs inline expansion at the call site.

```
#pragma inline_asm ( function-specification [, function-specification]... )Note
  function-specification: function-name [(size=numerical value)]
```

Note The outer parentheses can be omitted.

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.
Specifying (size = numerical value) does not affect the result of compilation.

Example

- C source

```
#pragma inline_asm      func_add
static int      func_add(int a, int b){
        add     r6, r7
        mov     r7, r10
}
void func(int *p){
        *p = func_add(10,20);
}
```

- Output codes

```
_func:
prepare r20, 0
mov     r6, r20
movea   0x0014, r0, r7
mov     10, r6
add     r6, r7
mov     r7, r10
```

(2) Notes for Use of #pragma inline_asm

- Specify #pragma inline_asm before the definition of the function body.
- Also generate external definitions of functions specified with #pragma inline_asm.
- If you use a register to guarantee the entrance and exit of an inline function with embedded assembly, you must save and restore this register at the beginning and end of the function.
- The compiler passes strings in #pragma inline_asm to the assembler as-is, without checking or modifying them.
- Only positive integer constants are specifiable for (size = numerical value). Specifying a floating-point number or value less than 0 will cause an error.
- If #pragma inline_asm is specified for a static function, then the function definition will be deleted after inline expansion.
- Assembly code is targeted by the preprocessor. Caution is therefore needed when using #define to define a macro with the same name as an instruction or register used in assembly language (e.g. "MOV" or "r5").
- Although it is possible to use comments starting with a hash ("#") in RH850 assembly language, if you use this comment, do not use # comments inside functions coded in assembly, because the preprocessor will interpret these as preprocessing directives.
- #pragma inline_asm cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma noinline, #pragma interrupt,
  #pragma block_interrupt, #pragma stack_protector,

- 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.
  - Write a code so that a label is expanded only in one location.
    - When calling an assembly-language function within the current source file, define the 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.

  The following shows a sample output code.
  - C source

```
#pragma inline_asm func1
static void func1(void)  /* When calling within the current file a */
{                        /* function that includes a label definition and */
                         /* is specified with inline_asm, code it as */
                         /* a static function. */
  .PUBLIC _label1
  add 1, r6
_label1:
  add -1, r6
}

void main(void) {
  func1();              /* Calls the function that includes a label */
                        /* definition and is specified with inline_asm. */
}

#pragma inline_asm func2
void func2(void)       /* When not calling within the current file a */
                       /* function that includes a label definition and */
                       /* is specified with inline_asm, code it as an */
                       /* external function. */
  .PUBLIC _label2
  add -1, r6
_label2:
  add 1, r6
}
```

  - Output assembly source code

```
_main:
  .stack _main = 0
  ._line_top inline_asm
  .PUBLIC _label1
  add 1, r6
_label1:
  add -1, r6
  ._line_end inline_asm
  jmp [r31]

_func2:
  ._line_top inline_asm
  .PUBLIC _label2
  add -1, r6
_label2:
  add 1, r6
  ._line_end inline_asm
  jmp [r31]
```

4.2.6.3 Inline expansion

The CC-RH allows inline expansion of each function. This section explains how to specify inline expansion.

(1) Inline Expansion

Inline expansion is used to expand the main body of a function at a location where the function is called. This decreases the overhead of function call and increases the possibility of optimization. As a result, the execution speed can be increased.
If inline expansion is executed, however, the object size increases.
Specify the function to be expanded inline using the #pragma inline directive.

```
#pragma inline ( function-name [, function-name]... )Note
```

Note The outer parentheses can be omitted.

Describe functions that are described in the C language. In the case of a function, "void func1() {}", specify "func1". Two or more function names can be specified with each delimited by "," (comma).

```
#pragma inline  func1, func2
void    func1() {...}
void    func2() {...}
void func(void) {
        func1();    /*function subject to inline expansion*/
        func2();    /*function subject to inline expansion*/
}
```

(2) Conditions of inline expansion

At least the following conditions must be satisfied for inline expansion of a function specified using the #pragma inline directive.
Inline expansion may not be executed even if the following conditions are satisfied, because of the internal processing of the CC-RH.

(a) A function that expands inline and a function that is expanded inline are described in the same file

A function that expands inline and a function that is expanded inline, i.e., a function call and a function definition must be in the same file. This means that a function described in another C source cannot be expanded inline. If it is specified that a function described in another C source is expanded inline, the CC-RH does not output a warning message and ignores the specification.

(b) The #pragma inline directive is described before function definition.

If the #pragma inline directive is described after function definition, the CC-RH outputs a warning message and ignores the specification. However, prototype declaration of the function may be described in any order. Here is an example.

Example

```
[Valid Inline Expansion Specification]
#pragma inline  func1, func2
void    func1();        /*prototype declaration*/
void    func2();        /*prototype declaration*/
void    func1() {...}   /*function definition*/
void    func2() {...}   /*function definition*/
```

```
[Invalid Inline Expansion Specification]
void    func1();        /*prototype declaration*/
void    func2();        /*prototype declaration*/
void    func1() {...}   /*function definition*/
void    func1() {...}   /*function definition*/
#pragma inline  func1, func2
```

(c) The number of arguments is the same between "call" and "definition" of the function to be expanded inline.

If the number of arguments is different between "call" and "definition" of the function to be expanded inline, the CC-RH ignores the specification.

(d) The types of return value and argument are the same between "call" and "definition" of the function to be expanded inline.
If the number of arguments is different between "call" and "definition" of the function to be expanded inline, the CC-RH ignores the specification. If the type of the argument is the integer type (including enum) or pointer-type, and in the same size, however, inline expansion is executed.

(e) The number of arguments of the function to be expanded inline is not variable.
If inline expansion is specified for a function with a variable arguments, the CC-RH outputs neither an error nor warning message and ignores the specification.

(f) Recursive function is not specified to be expanded inline.
If a recursive function that calls itself is specified for inline expansion, the CC-RH outputs neither an error nor warning message and ignores the specification. If two or more function calls are nested and if a code that calls itself exists, however, inline expansion may be executed.

(g) does not make calls via the addresses of functions for inline expansion.
If you call a function for inline expansion via its address, then the inline expansion specification will be ignored, without outputting an error or warning message.

(h) If you specify -Xmerge_files, then functions may be inlined even if they are not coded within the file.

(3) Functions for which inline expansion should be prevented
When using the -Oinline option, use #pragma noinline to prevent inline expansion of a specific function.

```
#pragma noinline ( function-name [, function-name]... )Note
```

Note The outer parentheses can be omitted.

- #pragma inline cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma noinline, #pragma interrupt,
  #pragma block_interrupt, #pragma stack_protector,
- #pragma noinline cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma noinline, #pragma interrupt,
  #pragma block_interrupt

(4) Examples of differences in inline expansion operation depending on option specification
Here are differences in inline expansion operation depending on whether the #pragma inline directive or an option is specified.

-Oinline=0	Inline expansion specification will be ignored, without outputting an error or warning message.
-Oinline=1	Inline expansion will be performed on functions specified for it.
-Oinline=2	Inline expansion will be performed on functions automatically, even if it is not specified. However, inline expansion will not be performed on functions specified as ineligible for inline expansion.
-Oinline=3	Inline expansion will be performed on functions automatically, even if it is not specified. However, inline expansion will not be performed on functions specified as ineligible for inline expansion.

(5) Sample inline expansion
Below is an example of inline expansion.

- C source

```
#pragma inline  (func)
static int      func(int a, int b)
{
        return  (a+b)/2;
}
int     x;
main()
{
        x = func (10, 20);
}
```

- Sample expansion

```
int     x;
main()
{
        int     func_result;
{
        int     a_1 = 10, b_1 = 20;
        func_result = (a_1+b_1)/2;
}
        x = func_result;
}
```

4.2.6.4 Controlling interrupt level

The CC-RH can manipulate the interrupts of the RH850 family as follows in a C source.

- By controlling interrupt level

- By enabling or disabling acknowledgment of maskable interrupts (by masking interrupts)

In other words, the interrupt control register can be manipulated.

(1) Controlling the interrupt priority level
For this purpose, the "__ set_il" function is used. Specify this function as follows to manipulate the interrupt priority level.

```
__set_il_rh(int interrupt-priority-level, void* address of interrupt control reg-
ister);
```

Integer values 1 to 16 can be specified as the interrupt priority level. With RH850, sixteen steps, from 0 to 15, can be specified as the interrupt priority level. To set the interrupt priority level to "5", therefore, specify the interrupt priority level as "6" by this function.

(2) Enable or disable acknowledgement of maskable interrupts (interrupt mask)
Specify the __ set_il function as follows to enable or disable acknowledgment of a maskable interrupt.

```
__set_il_rh(int enables/disables maskable interrupt, void* address of interrupt
control register);
```

Integer values -3 to 0 can be specified to enable or disable the maskable interrupt.

Set Value	Operation
0	Enables acknowledgement of maskable interrupt (unmasks interrupt).
-1	Disables acknowledgment of maskable interrupt (masks interrupt).
-2	To use direct branching (standard specification) as the interrupt vector method
-3	To use table lookup (extended specification) as the interrupt vector method

Caution This facility can be used only when the -Xcpu={g3m|g3k|g3mh|g3kh} option is specified. In other cases, a value should be directly written to the interrupt control register without using this facility.

4.2.6.5 Interrupt/Exception processing handler

The CC-RH can describe an "Interrupt handler" or "Exception handler" that is called if an interrupt or exception occurs. This section explains how to describe these handlers.

(1) Occurrence of interrupt/exception
If an interrupt or exception occurs in the RH850 family, the program jumps to a handler address corresponding to the interrupt or exception.
The arrangement of the handler addresses and the available interrupts vary depending on the device of the RH850. See the Relevant Device's User's Manual of each device for details.

How to describe interrupt servicing is explained specifically in "(3) Describing interrupt/exception handler".

(2) Processing necessary in case of interrupt/exception
If an interrupt/exception occurs while a function is being executed, interrupt/exception processing must be immediately executed. When the interrupt/exception processing is completed, execution must return to the function that was interrupted.
Therefore, the register information at that time must be saved when an interrupt/exception occurs, and the register information must be restored when interrupt/exception processing is complete.

(3) Describing interrupt/exception handler
The format in which an interrupt/exception handler is described does not differ from ordinary C functions, but the functions described in C must be recognized as an interrupt/exception handler by the CC-RH. With the CC-RH, an interrupt/exception handler is specified using the #pragma interrupt directive.

```
#pragma interrupt ( function-specification [, function-specification]... )Note
  function-specification: function-name [(interrupt specification [, interrupt
  specification]...)]
```

Note The outer parentheses can be omitted.

Describe functions that are described in the C language. In the case of a function, "void func1() {}", specify "func1".
The function exit code of an interrupt function is different from that of an ordinary function. You must therefore not call them in the same way as ordinary functions.

(a) interrupt specification
The following *interrupt specification* can be specified.

enable=	Specifies whether multiplex interrupts are enabled. This can be set to true, false, or manual. - true Output ei/di. Outputs code to save or restore eipc/eipsw. - false (default) Does not output ei/di. Does not output code to save or restore eipc/eipsw. - manual Does not output ei/di. Outputs code to save or restore eipc/eipsw.
priority= channel=	You can only specify one of either "priority" or "channel", but not both (writing both will cause a compilation error). - priority= Specifies the exception trigger. You can write only one of the following tokens. SYSERR/FETRAP/TRAP0/TRAP1/RIE/FPPNote 1/FPINote 1/FPINTNote 2/FPENote 3/ FXENote 3/UCPOP/MIP/MDP/PIE/MAE/FENMI/FEINT/EIINT_PRIORITYX (X is 0 -15) - channel= Specifies the interrupt channel. Select this if you are using the extended specification for interrupts. Generates code determining the EI compiler to be used. If you did not specify either "priority" or "channel", it will determine the EIINT.

fpu=	Specifies saving and restoration of fpepc/fpsr in the fpu context.[Note 4] This can be set to true, false, or auto. - true Saves and restores fpepc/fpsr. - false Does not save or restore fpepc/fpsr. - auto (default) Interpreted as true when the -Xfloat=fpu option is specified. Interpreted as false when -Xfloat=soft is specified.
fxu=[Note 3]	Specifies saving and restoration of fxsr/fxxp in the fxu context. This can be set to true, false, or auto. - true Saves and restores fxsr/fxxp. - false Does not save or restore fxsr/fxxp. - auto (default) Saves and restores fxsr/fxxp when the -Xfxu=on option is specified. Does not save or restore fxsr/fxxp when the -Xfxu=off option is specified.
callt=	Specifies saving/restoration of ctpc/ctpsw in the callt context. This can be set to true, false. - true (default) Saves and restore ctpc/ctpsw. - false Does not save or restore ctpc/ctpsw.
resbank[Note 3]	Outputs the resbank instruction to the exit code of a function. Part of the instruction string for saving context is not output. A warning is output in any of the following cases. - When the interrupt specification "fpu=false" was specified simultaneously - When -Xreg_mode=22 or -Xreg_mode=common was specified simultaneously - When -Xreserve_r2 was specified simultaneously - When -Xep=fix was specified simultaneously An error will occur in the following case. - When a token other than EIINT was specified simultaneously for the interrupt specification "priority="
param=	Specifies the method for receiving the value of an exception source register as a formal parameter. Specify the exception source register names in param=( ) for the number of formal parameters. One to four exception source register names can be specified, and they should be delimited with a comma (,). The specifiable exception source register names are as follows: eiic, feic, fpsr, fxsr[Note 3], fxxc[Note 3], fxxp[Note 3] An error will occur in any of the following cases. - When the number of specified exception source register names does not match the number of formal parameters - When the same exception source register name was specified more than once

Note 1. An error will occur if the -Xcpu=g3mh option is specified.

Note 2. An error will occur if the -Xcpu=g3mh option is not specified.

Note 3. An error will occur if the -Xcpu=g4mh option is not specified.

Note 4. If a CPU core that does not support an imprecise exception is specified, fpepc is not subject to saving or restoration. For details about the imprecise exception, see the user's manual of the device.

The parameter of the interrupt specification cannot be omitted.
For example, writing only "enable=" will cause a compilation error. The default interrupt specification signifies the behavior when individual interrupt specifications are not written.

(b) Definition of interrupt functions
The return type of an interrupt function should always be the void type.
For formal parameters of an interrupt function, a maximum of four can be written when the interrupt specification "param=" is specified and a maximum of one can be written when not specified.
The type of formal parameters should always be unsigned long.
When "param=" is not specified, the EIIC register value is stored in the formal parameter for an EI level exception and the FEIC register value is stored for any other exception.
When "param=" is specified, the value of each exception source register is stored in the corresponding formal parameter according to the specified contents.

Example

```
#pragma interrupt handler1 (priority=EIINT)
void handler1(unsigned long a) {  /* a = EIIC; */
    :
}

#pragma interrupt handler2
void handler2(unsigned long a) {  /* a = FEIC */
    :
}

#pragma interrupt handler3 (param=(eiic,feic,fpsr))
void handler3(unsigned long a, unsigned long b, unsigned long c) {
    /* a = EIIC, b = FEIC, c = FPSR */
    :
}
```

(c) Output code for EI level exception
The compiler inserts the following instructions at the entrance and exit of an EI level exception interrupt function.
EIINT and FPI are some of the main corresponding items.
However, this is not inserted into all interrupt functions. Necessary processing is output in accordance with user-defined #pragma statements, compiler options, etc.

<1> [Entrance code of interrupt functions]
(1) Allocates stack area for saving context
(2) Saves Caller-Save register used in interrupt function
(3) Saves EIPC and EIPSW
(4) If the function has a formal parameter, set EIIC to R6
(5) Enables multiplex interrupts
(6) Saves WCTPC and CTPSW
(7) Saves WFPEPC and FPSR

<2> [Exit code of interrupt functions]
(8) Sets imprecise interrupt standby
(9) Restorse FPEPC and FPSR
(10) Restores CTPC and CTPSW
(11) Disables multiplex interrupts
(12) Restores EIPC and EIPSW
(13) Restores Caller-Save register used in interrupt function
(14) Frees stack area for saving context
(15) eiret

Below is a specific example of the output code. Numbers (1) to (15) in the code correspond to the numbered actions above.
Note that the instructions in the output code will not necessarily be identical to this example. The instructions, general-purpose registers, and other details may differ from this example.

Example 1. Sample1: output of EI level exception

```
#pragma interrupt func1(enable=true, callt=true, fpu=true)
void func1(unsigned long eiic)
{
        User-coded processing;
}
```

```
_func1:
    movea   -0x00000038, r3, r3 ; (1)
    st23.dw r6, 0x00000030[r3]  ; (2)
    stsr    0, r6               ; (3)
    stsr    1, r7               ; (3)
    st23.dw r6, 0x00000028[r3]  ; (3)
    stsr    13, r6              ; (4)
    ei                          ; (5)
    st23.dw r4, 0x00000020[r3]  ; (2)
    st23.dw r8, 0x00000018[r3]  ; (2)
    st23.dw r10, 0x00000010[r3] ; (2)
    stsr    16, r8              ; (6)
    stsr    17, r9              ; (6)
    st23.dw r8, 0x00000008[r3]  ; (6)
    stsr    7, r8               ; (7)
    stsr    6, r9               ; (7)
    st23.dw r8, 0x00000000[r3]  ; (7)
    prepare ....                ; Saves Callee-Save register
        :                       ; User-coded processing
    dispose ....                ; Restores Callee-Save register
    synce                       ; (8)
    ld23.dw 0x00000000[r3], r8  ; (9)
    ldsr    r8, 7               ; (9)
    ldsr    r9, 6               ; (9)
    ld23.dw 0x00000008[r3], r8  ; (10)
    ldsr    r8, 16              ; (10)
    ldsr    r9, 17              ; (10)
    ld23.dw 0x00000010[r3], r10 ; (13)
    ld23.dw 0x00000018[r3], r8  ; (13)
    ld23.dw 0x00000020[r3], r4  ; (13)
    di                          ; (11)
    ld23.dw 0x00000028[r3], r6  ; (12)
    ldsr    r6, 0               ; (12)
    ldsr    r7, 1               ; (12)
    ld23.dw 0x00000030[r3], r6  ; (13)
    movea   0x00000038, r3, r3  ; (14)
    eiret                       ; (15)
```

Compiler embeds entrance code into beginning of interrupt function

Interrupt processing coded by user

Compiler embeds exit code into end of interrupt function

Example 2. Sample2: output of EI level exception
If there are no formal parameters, and interrupt multiplexing is set to manual (enable=manual)

```
#pragma interrupt func1(enable=true, callt=true, fpu=true)
void func1(unsigned long eiic)
 {
        User-coded processing;
}
```

```
_func1:
    movea   -0x00000038, r3, r3 ; (1)
    st23.dw r6, 0x00000030[r3]  ; (2)
    stsr    0, r6               ; (3)
    stsr    1, r7               ; (3)
    st23.dw r6, 0x00000028[r3]  ; (3)
    st23.dw r4, 0x00000020[r3]  ; (2)
    st23.dw r8, 0x00000018[r3]  ; (2)
    st23.dw r10, 0x00000010[r3] ; (2)
    stsr    16, r8              ; (6)
    stsr    17, r9              ; (6)
    st23.dw r8, 0x00000008[r3]  ; (6)
    stsr    7, r8               ; (7)
    stsr    6, r9               ; (7)
    st23.dw r8, 0x00000000[r3]  ; (7)
    prepare ....                ; Saves Callee-Save register
        :                       ; User-coded processing
    dispose ....                ; Restores Callee-Save register
    ld23.dw 0x00000000[r3], r8  ; (9)
    ldsr    r8, 7               ; (9)
    ldsr    r9, 6               ; (9)
    ld23.dw 0x00000008[r3], r8  ; (10)
    ldsr    r8, 16              ; (10)
    ldsr    r9, 17              ; (10)
    ld23.dw 0x00000010[r3], r10 ; (13)
    ld23.dw 0x00000018[r3], r8  ; (13)
    ld23.dw 0x00000020[r3], r4  ; (13)
    ld23.dw 0x00000028[r3], r6  ; (12)
    ldsr    r6, 0               ; (12)
    ldsr    r7, 1               ; (12)
    ld23.dw 0x00000030[r3], r6  ; (13)
    movea   0x00000038, r3, r3  ; (14)
    eiret                       ; (15)
```

Compiler embeds entrance code into beginning of interrupt function

Interrupt processing coded by user

Compiler embeds exit code into end of interrupt function

(d) Output code for FE level exception
The compiler inserts the following instructions at the entrance and exit of an FE level exception interrupt function. FEINT and PIE are some of the main corresponding items.
However, this is not inserted into all interrupt functions. Necessary processing is output in accordance with user-defined #pragma statements, compiler options, etc.

<1> [Entrance code of interrupt functions]
(1) Allocates stack area for saving context
(2) Saves all Caller-Save register used in interrupt function
(3) If the function has a formal parameter, sets FEIC to R6
(4) Saves CTPC and CTPSW
(5) Saves FPEPC and FPSR

<2> [Exit code of interrupt functions]
(6) Restores FPEPC and FPSR
(7) Restores CTPC and CTPSW
(8) Restores all Caller-Save register used in interrupt function
(9) Frees stack area for saving context
(10) feret

Below is a specific example of the output code. Numbers (1) to (10) in the code correspond to the numbered actions above.
Note that the instructions in the output code will not necessarily be identical to this example. The instructions, general-purpose registers, and other details may differ from this example.

Example Sample output of FE level exception

```
#pragma interrupt func1(priority=feint, callt=true, fpu=true)
void func1(unsigned long feic)
{
        User-coded processing;
}
```

```
_func1:
    movea   -0x00000030, r3, r3 ; (1)
    st23.dw r4, 0x00000028[r3]  ; (2)
    st23.dw r6, 0x00000020[r3]  ; (2)
    st23.dw r8, 0x00000018[r3]  ; (2)
    st23.dw r10, 0x00000010[r3] ; (2)
    stsr    14, r6              ; (3)
    stsr    16, r8              ; (4)
    stsr    17, r9              ; (4)
    st23.dw r8, 0x00000008[r3]  ; (4)
    stsr    7, r8               ; (5)
    stsr    6, r9               ; (5)
    st23.dw r8, 0x00000000[r3]  ; (5)
    prepare ....                ; Saves Callee-Save registe
        :                       ; User-coded processing
    dispose ....                ; Restores Callee-Save register
    ld23.dw 0x00000000[r3], r8  ; (6)
    ldsr    r8, 7               ; (6)
    ldsr    r9, 6               ; (6)
    ld23.dw 0x00000008[r3], r8  ; (7)
    ldsr    r8, 16              ; (7)
    ldsr    r9, 17              ; (7)
    ld23.dw 0x00000010[r3], r10 ; (8)
    ld23.dw 0x00000018[r3], r8  ; (8)
    ld23.dw 0x00000020[r3], r6  ; (8)
    ld23.dw 0x00000028[r3], r4  ; (8)
    movea   0x00000030, r3, r3  ; (9)
    feret                       ; (10)
```

Compiler embeds entrance code into beginning of FEINT

FEINT processing coded by user

Compiler embeds exit code into end of FEIN

(e) Output code for FE level exception (cannot recover/restore)
The compiler inserts the following instructions at the entrance and exit of an FE level exception (cannot recover/restore) interrupt function. FENMI and SYSERR are some of the main corresponding items.

<1> [Entrance code of interrupt functions]
(1) If the function has a formal parameter, sets FEIC to R6
Nothing is output if the function does not have any parameters.

<2> [Exit code of interrupt functions]
None

Remark No saving or restoration of context is output.
Code the save and restore the Callee-Save register is also not output.
Take appropriate measures in the user program, such as calling abort() in the function to terminate the program.

Below is a specific example of the output code. Numbers (1) in the code correspond to the numbered actions above.

Example Sample output of FE level exception (cannot recover/restore)

```
#pragma interrupt func1(priority=fenmi)
void func1(unsigned long feic)
{
        User-coded processing;
}
```

```
_func1:

    stsr    14, r6                  ; (1)
        :
        :                           ; User-coded processing
        :
```

Compiler embeds entrance code into beginning of FENMI

FENMI processing coded by user

No code is output at the end

(f) Output code for EI level exception when resbank is specified
The compiler inserts the following instructions at the entrance and exit of an EI level exception interrupt function for which resbank is specified.
The instruction string differs from that output for a normal EI level exception interrupt function in the following points.

- There is no instruction string for saving the context that is automatically saved by the register bank facility.
- The resbank instruction is output instead of the instruction string for restoring the above context.

Note that information of the context to be automatically saved by the register bank facility is determined from the specified -Xresbank_mode option.
These are not always inserted in all interrupt functions, and the necessary processing is output in response to the #pragma directives written by the user or the compile options.

When resbank is not specified	When resbank is specified (-Xresbank_mode=0)

_handler:
movea 0xFFFFFFA8, r3, r3
st.w r1, 0x14[r3] ; save r1
st.w r2, 0x18[r3] ; save r2
st.w r5, 0x1C[r3] ; save r5
st23.dw r6, 0x20[r3] ; save r6,r7
st23.dw r8, 0x28[r3] ; save r8,r9
st23.dw r10, 0x30[r3] ; save r10,r11
st23.dw r12, 0x38[r3] ; save r12,r13
st23.dw r14, 0x40[r3] ; save r14,r15
st23.dw r16, 0x48[r3] ; save r16,r17
st23.dw r18, 0x50[r3] ; save r18,r19
stsr 0x11, r9, 0x00
stsr 0x10, r8, 0x00
st23.dw r8, 0x00[r3] ; save CTPC,CTPSW
stsr 0x06, r8, 0x00
st.w r8, 0x08[r3] ; save FPSR
stsr 0x06, r8, 0x0A
stsr 0x0D, r9, 0x0A
st23.dw r8, 0x0C[r3] ; save FXSR,FXXP
prepare 0x01, 0x00 ; save r31
jarl _sub, r31
dispose 0x00, 0x01 ; restore r31
ld23.dw 0x0C[r3], r8
ldsr r9, 0x0D, 0x0A ; restore FXXP
ldsr r8, 0x06, 0x0A ; restore FXSR
ld.w 0x08[r3], r8
ldsr r8, 6 ; restore FPSR
ld23.dw 0x00[r3], r8
ldsr r8, 0x10, 0x00 ; restore CTPC
ldsr r9, 0x11, 0x00 ; restore CTPSW
ld23.dw 0x50[r3], r18 ; restore r18,r19
ld23.dw 0x48[r3], r16 ; restore r16,r17
ld23.dw 0x40[r3], r14 ; restore r14,r15
ld23.dw 0x38[r3], r12 ; restore r12,r13
ld23.dw 0x30[r3], r10 ; restore r10,r11
ld23.dw 0x28[r3], r8 ; restore r8,r9
ld23.dw 0x20[r3], r6 ; restore r6,r7
ld.w 0x1C[r3], r5 ; restore r5
ld.w 0x18[r3], r2 ; restore r2
ld.w 0x14[r3], r1 ; restore r1
movea 0x58, r3, r3
eiret
	_handler:
	movea 0xFFFFFFF0, r3, r3
	
	stsr 0x11, r9, 0x00
	stsr 0x10, r8, 0x00
	st23.dw r8, 0x00[r3] ; save CTPC,CTPSW
	
	stsr 0x06, r8, 0x0A
	stsr 0x0D, r9, 0x0A
	st23.dw r8, 0x08[r3] ; save FXSR,FXXP
	prepare 0x01, 0x00 ; save r31
	jarl _sub, r31
	dispose 0x00, 0x01 ; restore r31
	ld23.dw 0x08[r3], r8
	ldsr r9, 0x0D, 0x0A ; restore FXXP
	ldsr r8, 0x06, 0x0A ; restore FXSR
	
	ld23.dw 0x00[r3], r8
	ldsr r8, 0x10, 0x00 ; restore CTPC
	ldsr r9, 0x11, 0x00 ; restore CTPSW
	resbank ; restore r1-r19,r30,EIIC,FPSR
	
	eiret

[Remark]
Even though the interrupt specification "resbank" is used, a code for setting a value to RBCR0 is not generated. The value must be directly set with the user program.

(4) Notes on describing interrupt/exception handler

- #pragma interrupt cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma noinline, #pragma interrupt,
  #pragma block_interrupt

4.2.6.6 Disabling or enabling maskable interrupts

The CC-RH can disable the maskable interrupts in a C source.
This can be done in the following two ways.

- Locally disabling interrupt in function
- Disabling interrupts in entire function

(1) Locally disabling interrupt in function
The "di instruction" and "ei instruction" of the assembler instruction can be used to disable an interrupt locally in a function described in C language. However, the CC-RH has functions that can control the interrupts in a C language source.

Table 4.19 Interrupt Control Function

Interrupt Control Function	Operation	Processing by CC-RH
__DI	Disables the acceptance of all maskable interrupts.	Generates di instruction.
__EI	Enables the acceptance of all maskable interrupts.	Generates ei instruction.

Example How to use the __DI and __EI functions and the codes to be output are shown below.

- C source

```
void func1(void) {
          :
        __DI();
        /*Describe processing to be performed with interrupt disabled.*/
        __EI();
          :
}
```

- Output codes

```
_func1:
        -- prologue code
          :
        di
        -- processing to be performed with interrupt disabled
        ei
          :
        -- epilogue code
        jmp     [lp]
```

(2) Disabling interrupts in entire function
The CC-RH has a "#pragma block_interrupt" directive that disables the interrupts of an entire function.
This directive is described as follows.

```
#pragma block_interrupt ( function-name [, function-name]... )Note
```

Note The outer parentheses can be omitted.

Describe functions that are described in the C language. In the case of a function, "void func1() {}", specify "func1".
The interrupt to the function specified by "function-name" above is disabled. As explained in "(1) Locally disabling interrupt in function", __ DI()" can be described at the beginning of a function and "__ EI()", at the end. In this case, however, an interrupt to the prologue code and epilogue code output by the CC-RH cannot be disabled or enabled, and therefore, interrupts in the entire function cannot be disabled.
Using the #pragma block_interrupt directive, interrupts are disabled immediately before execution of the prologue code, and enabled immediately after execution of the epilogue code. As a result, interrupts in the entire function can be disabled.

Example How to use the #pragma block_interrupt directive and the code that is output are shown below.

- C source

```
#pragma block_interrupt func1
void func1(void) {
          :
        /*Describe processing to be performed with interrupt disabled.*/
          :
}
```

- Output codes

```
_func1:
        di
        -- prologue code
          :
        -- processing to be performed with interrupt disabled
          :
        -- epilogue code
        ei
        jmp     [lp]
```

(3) Notes on disabling interrupts in entire function

Note the following points when disabling interrupts in an entire function.

- If the following functions are called in a function in which an interrupt is disabled, the interrupt is enabled when execution has returned from the call.
  - Function specified by #pragma block_interrupt.
  - Function that disables interrupt at the beginning and enables interrupt at the end.
- Describe the #pragma block_interrupt directive before the function definition in the same file; otherwise an error occurs during compilation.
- However, the order of prototype declaration of a function is not affected.
- A code that manipulates the ep flag (that indicates exception processing is in progress) in the program status word (PSW) is not output even if #pragma block_interrupt is specified.
- #pragma block_interrupt cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma noinline, #pragma interrupt,
  #pragma block_interrupt

4.2.6.7 Intrinsic functions

In the CC-RH, some of the assembler instructions can be described in C source as "Intrinsic functions". However, it is not described "as assembler instruction", but as a function format set in the CC-RH.

If a parameter is specified whose type cannot be implicitly converted to that of the parameter of the intrinsic function, then an warning is output, and it is treated as an intrinsic function.

See the user's manual of the relevant device for operation when a register number not available in hardware is specified for ldsr/stsr.

The instructions that can be described as functions are as follows.

Table 4.20 Assembler Instruction (1)

Assembler Instruction	Function	Intrinsic Function
di	Interrupt control	void __DI(void);
ei		void __EI(void);

-	Interrupt-priority-level control- Note 1	void __set_il_rh(long NUM, void* ADDR Note 2);
		
		- NUM : 1 - 16
		movhi highw1(ADDR), r0, rX
		ld.b loww(ADDR)[rX], rY
		andi 0x00F0, rY, rY
		ori (Priority - 1), rY, rY
		st.b loww(ADDR)[rX]
		- NUM : 0
		movhi highw1(ADDR), r0, rX
		clr1 7, loww(ADDR)[rX]
		- NUM : -1
		movhi highw1(ADDR), r0, rX
		set1 7, loww(ADDR)[rX]
		- NUM : -2
		movhi highw1(ADDR), r0, rX
		clr1 6, loww(ADDR)[rX]
		- NUM : -3
		movhi highw1(ADDR), r0, rX
		set1 6, loww(ADDR)[rX]
		- NUM : No greater than 4 and no less than 17
		Out-of-range error

nop	No operation	void __nop(void);
halt	Stops the processor	void __halt(void);
satadd	Saturated addition	long __satadd(long a, long b);
satsub	Saturated subtraction	long __satsub(long a, long b);
bsh	Halfword data byte swap	long __bsh(long a);
bsw	Word data byte swap	long __bsw(long a);
hsw	Word data halfword swap	long __hsw(long a);
mul	Instruction that assigns higher 32 bits of signed 64-bit multiplication result to variable	long __mul32(long a, long b);

Assembler Instruction	Function	Intrinsic Function
mulu	Instruction that assigns higher 32 bits of unsigned 64-bit multiplication result to variable	unsigned long __mul32u(unsigned long a, unsigned long b);
sch0l	Bit (0) search from MSB side	long __sch0l(long a);
sch0r	Bit (0) search from LSB side	long __sch0r(long a);
sch1l	Bit (1) search from MSB side	long __sch1l(long a);
sch1r	Bit (1) search from LSB side	long __sch1r(long a);
ldsr	Loads to system register	void __ldsr(long regID[Note 3], unsigned long a);
ldsr	Loads to system register	void __ldsr_rh(long regID[Note 3], long selID[Note 3], unsigned long a);
stsr	Stores contents of system register	unsigned long __stsr(long regID[Note 3]);
stsr	Stores contents of system register	unsigned long __stsr_rh(long regID[Note 3], long selID[Note 3]);
caxi	Compare and Exchange	long __caxi(long *a, long b, long c);
clr1	Bit clear	void __clr1(unsigned char *a, long bit);
set1	Bit set	void __set1(unsigned char *a, long bit);
not1	Bit not	void __not1(unsigned char *a, long bit);
ldl.w[Note 4]	Atomic load.	long __ldlw(long *a);
stc.w[Note 4]	Store	void __stcw(long *a, long b); [V1.04.00 or earlier] long __stcw(long *a, long b); [V1.05.00 or later]
synce	Exception synchronization	void __synce(void);
synci	Instruction pipeline synchronization	void __synci(void);
syncm	Memory synchronization	void __syncm(void);
syncp	Pipeline synchronization	void __syncp(void);
dbcp	Debug checkpoint	void __dbcp(void);
dbpush	Debug push	void __dbpush(long regID1, long regID2);
dbtag	Debug tag	void __dbtag(long a);

Note 1. The __set_il_rh function can be used only when the –Xcpu={g3m|g3k|g3mh|g3kh} option is specified.

Note 2. For ADDR, specify the address of the interrupt control register.

Note 3. Specified the system register number (0 to 31) in regID and 0 to 31 in selID.

Note 4. A warning is output when the -Xcpu=g3k option is used.

Caution Even if a function is defined with the same name as an intrinsic function, it cannot be used.
If an att isempt made to call such a function, processing for the intrinsic function provided by the compiler takes precedence.

When the -Xcpu=g4mh option is specified, the following intrinsic functions can also be used.

Table 4.21 Assembler Instruction (2)

Assembler Instruction	Function	Intrinsic Function
clip.b	Conversion of signed word data to byte data with saturation	`long __clipb(long a);`
clip.bu	Conversion of unsigned word data to byte data with saturation	`unsigned long __clipbu(unsigned long a);`
clip.h	Conversion of signed word data to halfword data with saturation	`long __cliph(long a);`
clip.hu	Conversion of unsigned word data to halfword data with saturation	`unsigned long __cliphu(unsigned long a);`
ldl.bu	Load which starts atomic byte data manipulation	`long __ldlbu(unsigned char* a);`
ldl.hu	Load which starts atomic halfword data manipulation	`long __ldlhu(unsigned short* a);`
stc.b	Store on the condition that byte data manipulation has been completed atomically	`long __stcb(unsigned char* a, unsigned char b);`
stc.h	Store on the condition that halfword data manipulation has been completed atomically	`long __stch(unsigned short* a, unsigned short b);`

4.2.6.8 Structure type packing

In the CC-RH, the alignment of structure members can be specified at the C language level. This function is equivalent to the -Xpack option, however, the structure type packing directive can be used to specify the alignment value in any location in the C source.

Caution The data area can be reduced by packing a structure type, but the program size increases and the execution speed is degraded.

(1) Format of structure type packing
The structure type packing function is specified in the following format.

```
#pragma pack ({1|2|4 })Note
```

Note The outer parentheses can be omitted.

#pragma pack changes to an alignment value of the structure member upon the occurrence of this directive. The numeric value is called the packing value and the specifiable numeric values are 1, 2, 4. When the numeric value is not specified, the setting is the default alignment.Since this directive becomes valid upon occurrence, several directives can be described in the C source.

Example

```
#pragma pack 1  /*structure member aligned using 1-byte alignment*/
struct  TAG {
        char    c;
        int     i;
        short   s;
};
```

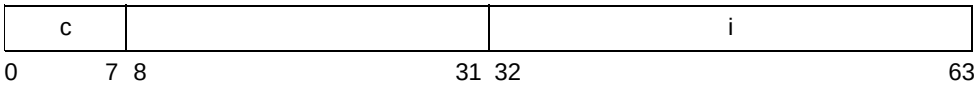
(2) Rules of structure type packing

The structure members are aligned in a form that satisfies the condition whereby members are aligned according to whichever is the smaller value: the structure type packing value or the member's alignment value.
For example, if the structure type packing value is 2 and member type is int type, the structure members are aligned in 2-byte alignment.

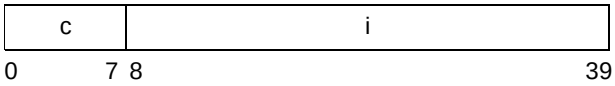
Example

```
struct  S {
        char    c;  /*satisfies 1-byte alignment condition*/
        int     i;  /*satisfies 4-byte alignment condition*/
};

#pragma pack 1
struct  S1 {
        char    c;  /*satisfies 1-byte alignment condition*/
        int     i;  /*satisfies 1-byte alignment condition*/
};

#pragma pack 2
struct  S2 {
        char    c;  /*satisfies 1-byte alignment condition*/
        int     i;  /*satisfies 2-byte alignment condition*/
};

struct  S   sobj;   /*size of 8 bytes*/
struct  S1  s1obj;  /*size of 5 bytes*/
struct  S2  s2obj;  /*size of 6 bytes*/
```

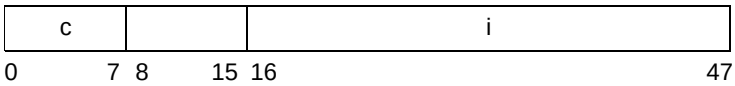
sobj



s1obj



s2obj



(3) Union

A union is treated as subject to packing and is handled in the same manner as structure type packing.

Example 1.

```
union   U {
        char c;
        int i;
};

#pragma pack 1
union   U1 {
        char c;
        int i;
};

#pragma pack 2
union   U2 {
        char c;
        int i;
};

union   U   uobj;   /*size of 4 bytes*/
union   U1  u1obj;  /*size of 4 bytes*/
union   U2  u2obj;  /*size of 4 bytes*/
```

Example 2.

```
union   U {
        int i:7;
};

#pragma pack 1
union   U1 {
        int i:7;
};

#pragma pack 2
union   U2 {
        int i:7;
};

union   U   uobj;   /*size of 4 bytes*/
union   U1  u1obj;  /*size of 1 byte*/
union   U2  u2obj;  /*size of 2 bytes*/
```

(4) Bit field
Data is allocated to the area of the bit field element as follows.

(a) When the structure type packing value is equal to or larger than the alignment condition value of the member type
Data is allocated in the same manner as when the structure type packing function is not used. That is, if the data is allocated consecutively and the resulting area exceeds the boundary that satisfies the alignment condition of the element type, data is allocated from the area satisfying the alignment condition.

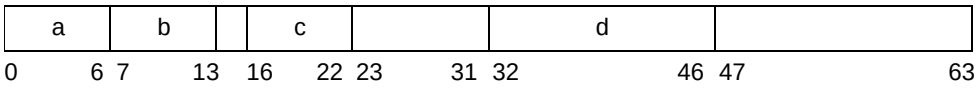
(b) When the structure type packing value is smaller than the alignment condition value of the element type

- If data is allocated consecutively and results in the number of bytes including the area becoming larger than the element type
  The data is allocated in a form that satisfies the alignment condition of the structure type packing value.
- Other conditions
  The data is allocated consecutively.

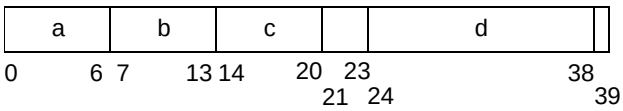
Example

```
struct  S {
        short   a:7;    /*0 to 6th bit*/
        short   b:7;    /*7 to 13th bit*/
        short   c:7;    /*16 to 22nd bit (aligned to 2-byte boundary)*/
        short   d:15;   /*32 to 46th bit (aligned to 2-byte boundary)*/
} sobj;

#pragma pack 1
struct  S1 {
        short   a:7;    /*0 to 6th bit*/
        short   b:7;    /*7 to 13th bit*/
        short   c:7;    /*14 to 20th bit*/
        short   d:15;   /*24 to 38th bit  (aligned to byte boundary)*/
} s1obj;
```

sobj



s1obj



(5) Alignment condition of top structure object
The alignment condition of the top structure object is the smaller of the alignment value and packing value of the structure object.

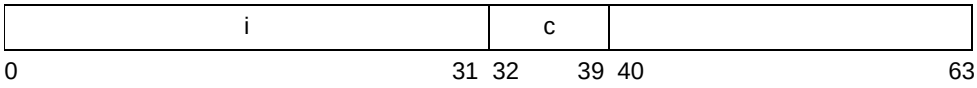
(6) Size of structure objects

Perform packing so that the size of structure objects becomes a multiple value of whichever is the smaller value: the structure alignment condition value or the structure packing value.

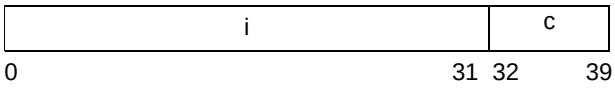
Example 1.

```
struct  S {
        int     i;
        char    c;
};

#pragma pack 1
struct  S1 {
        int     i;
        char    c;
};

#pragma pack 2
struct  S2 {
        int     i;
        char    c;
};

struct  S   sobj;   /*size of 8 bytes*/
struct  S1  s1obj;  /*size of 5 bytes*/
struct  S2  s2obj;  /*size of 6 bytes*/
```

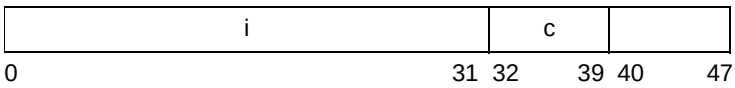
sobj



s1obj



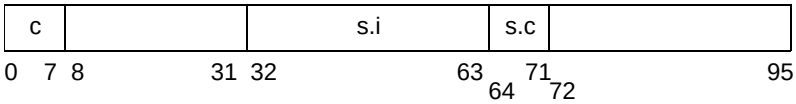
s2obj



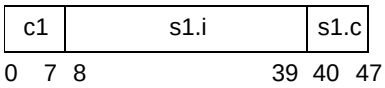
Example 2.

```
struct  S {
        int     i;
        char    c;
};
struct  T {
        char    c;
        struct  S   s;
};

#pragma pack 1
struct  S1 {
        int     i;
        char    c;
};
struct  T1 {
        char    c;
        struct  S1  s1;
};

#pragma pack 2
struct  S2 {
        int     i;
        char    c;
};
struct  T2 {
        char    c;
        struct  S2  s2;
};

struct  T   tobj;   /*size of 12 bytes*/
struct  T1  t1obj;  /*size of 6 bytes*/
struct  T2  t2obj;  /*size of 8 bytes*/
```

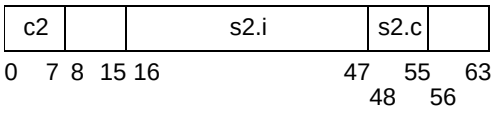
tobj



t1obj



t2obj



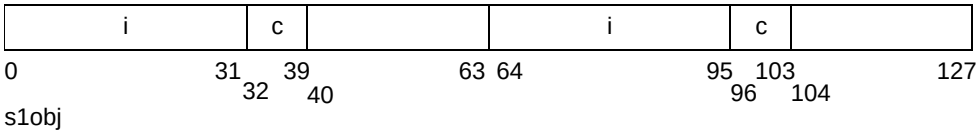
(7) Size of structure array

The size of the structure object array is a value that is the sum of the number of elements multiplied to the size of structure object.

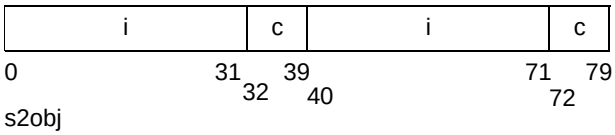
Example

```
struct  S {
        int     i;
        char    c;
};

#pragma pack 1
struct  S1 {
        int     i;
        char    c;
};

#pragma pack 2
struct  S2 {
        int     i;
        char    c;
};

struct  S   sobj[2];    /*size of 16 bytes*/
struct  S1  s1obj[2];   /*size of 10 bytes*/
struct  S2  s2obj[2];   /*size of 12 bytes*/
```

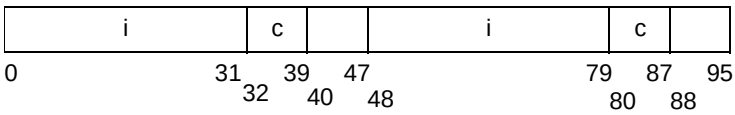
sobj



s1obj



s2obj



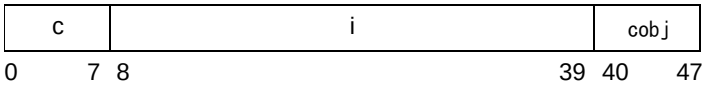
(8) Area between objects

For example, sobj.c, sobj.i, and cobj may be allocated consecutively without a gap in the following source program (the allocation order of sobj and cobj is not guaranteed).

Example

```
#pragma pack 1
struct  S {
        char    c;
        int     i;
} sobj;
char    cobj;
```

sobj, cobj



(9) Notes concerning structure packing function

(a) Specification of the -Xpack option and #pragma pack directive at the same time

If the -Xpack option is specified when structure packing is specified with the #pragma pack directive in the C source, the specified option value is applied to all the structures until the first #pragma pack directive appears. After this, the value of the #pragma pack directive is applied.

If you subsequently write #pragma pack (no value), then the value specified with this option is applied following that line.

Example When -Xpack=4 is specified

```
struct  S2 {...};   /*Packing value is specified as 4 in option.
                      -Xpack=4 option is valid: packing value is 4.*/

#pragma pack 2      /*Packing is specified as 2 in #pragma directive
struct  S1 {...};     pragma pack(2) is valid: packing value is 2.*/

#pragma pack        /*No specification of packing value with #pragma directive
struct  S2_2 {...};   -Xpack=4 option is valid: packing value is 4.*/
```

(b) Structure packing value and alignment value of members

Structure members are arranged so that the alignment conditions match the smaller of the structure's packing value and the members' alignment value. For example, if the structure's packing value is 2, and a member type is long, then it is ordered to meet the 2-byte alignment condition.

Example

```
struct S {
        char    c;      /*Meets 1-byte alignment condition*/
        long    i;      /*Meets 4-byte alignment condition*/
};

#pragma pack(1)
struct S1 {
        char    c;      /*Meets 1-byte alignment condition*/
        long    i;      /*Meets 1-byte alignment condition*/
};

#pragma pack(2)
struct S2 {
        char    c;      /*Meets 1-byte alignment condition*/
        long    i;      /*Meets 2-byte alignment condition*/
};

struct S        sobj;   /*Size 8 bytes*/
struct S1       s1obj;  /*Size 5 bytes*/
struct S2       s2obj;  /*Size 6 bytes*/
```

(c) Nested #pragma pack specification
Specify nested #pragma pack specifications for a structure as follows.
A warning is output for structure or union members with different alignment.
The alignment of members generating the warning will be in accordance with the #pragma pack statements in the source code.

Example

```
#pragma pack 1
struct ST1
{
        char    c;
#pragma pack 4
        struct ST4      //size=8, align=4 (Type is 4)
        {
            char    c;  //offset=1
            short   s;  //offset=3
            int     i;  //offset=5
        } st4;          //size=8, align=1 (1, because this is an ST1 member)
                        //Warning at location of member st4
        int i;
} st1;                  //size=13, align=1
```

[Caution]

When -Xpack=1 or 2 or #pragma pack 1 or 2 is specified for a structure or union, its members cannot be accessed using a pointer.

```
#pragma   pack  1
struct st {
    char  x;
    int  y;
} ST;
int *p = &ST.y; /* The ST.y address may be an odd value. */
void func(void){
    ST.y =1;    /* Can be accessed correctly. */
    *p = 1;     /* Cannot be accessed correctly in some cases. */
}
```

4.2.6.9 Bit field assignment

CC-RH can switch the order of a bit field.

(1) Format for specifying bit field assignment
Specify bit field assignment using the following format.

```
#pragma bit_order [{left|right}]
```

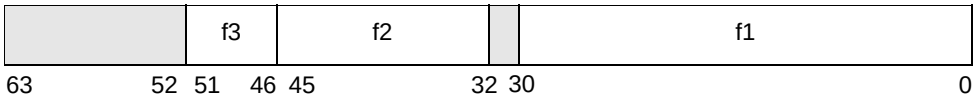
If left is specified, then members are assigned from the MSB; if right is specified, then they are assigned from the LSB.

Example 1. The default is right.

```
#pragma bit_order right
struct {
        unsigned int    f1:30;
        int             f2:14;
        unsigned int    f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.9 Internal Representation of Bit Field

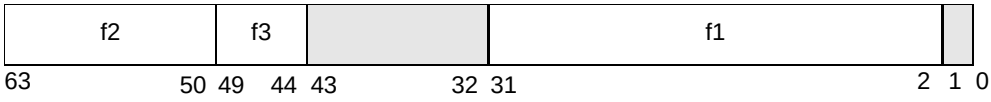


Example 2.

```
#pragma bit_order left
struct {
        unsigned int    f1:30;
        int             f2:14;
        unsigned int    f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.10 Internal Representation of Bit Field

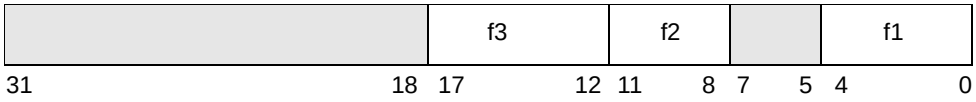


Example 3.

```
#pragma bit_order right
struct {
    int             f1:5;
    char            f2:4;
    int             f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.11 Internal Representation of Bit Field

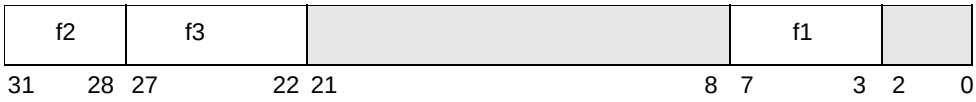


Example 4.

```
#pragma bit_order left
struct {
    int             f1:5;
    char            f2:4;
    int             f3:6;
} flag;
```

The internal representation for the bit field in the above example is as follows.

Figure 4.12 Internal Representation of Bit Field



4.2.6.10 Core number specification (for a multi-core device)

The core number specification function enables selection of data allocation area (the local memory in a specified core or the global memory shared by all cores) or selection of the core for function execution (a specified core or any core) when a multi-core device is used.

This function is specified by a combination of the #pragma directive described below and link options.

For example, to allocate variable x (assumed to be allocated to a data section) to the local memory for core number 1, specify as follows.

(a) Specify a pragma directive as follows before the first definition or declaration of variable x in the file:

```
#pragma pmodule pm1
```

This makes the compiler and assembler allocate variable x to section .data.pm1.

(b) Specify the following link option:

```
-start=.data.pm1/fe8f0000
```

This makes the linker allocate section .data.pm1 to the local memory for core number 1 (This example assumes 0xfe8f0000 as an address in the local memory for core number 1).

Specifying core numbers for variables or functions has the following merits.

- When a core number is added to each section name, the user can manage the correspondence between cores and variables or functions; that is, which variable is allocated to the local memory of which core and which function is executed in which core.
  This information can be viewed through CS+.
- As core numbers are added to all section names including the default section names, the user does not need to change the section names for every core.

(1) Format for specifying core number

Specify a core number for a multi-core device in the following format.

```
#pragma pmodule pm-specification
```

This pragma directive is valid only when the -Xmulti_level=1 option is specified. If the -Xmulti_level=1 option is not specified, a warning is output and the core number specification is ignored.

The following table shows the available pm specification forms and the names of the corresponding allocated sections.
Only pm1 to pm255 or cmn can be written as pm specification. For a variable or a function with pm specification, a period (.) and the string used for pm specification is added at the end of the allocated section name.

-Xmulti_level Value	pm Specification Value	Meaning	Name of Allocation Section
1	None	Default (cmn) is specified.	***.cmn
	cmn	- For data Allocated to the global shared memory used in common for all cores. - For a function The function can be executed in any core	***.cmn
	pm*N*	Data or function for core *N*	***.pm*N*

-Xmulti_level Value	pm Specification Value	Meaning	Name of Allocation Section
0	A warning message is output and the core number specification is ignored.		*** (No string is added at the end of the section name.)

- If a pm specification other than the specifications below is made, a compilation error will occur.
  - cmn, pm1 to pm255 [V1.07.00 or earlier]
  - cmn, pm0 to pm255 [V2.00.00 or later]
- #pragma pmodule is applied to all static variable declarations, function declarations, and string literals that appear after the #pragma pmodule declaration line.[Note]

  Note This directive is not applied to the initial value data of structure-type, union-type, or array-type auto variables.

- The #pragma pmodule directive adds the string described above to both the default section names and user-specified section names.

  Example .data -> .data.pm1
  mydata.data -> mydata.data.pm1

- When there are variables or functions with different #pragma pmodule specifications other than cmn within a single translation unit, an error will occur.
- The possible combinations for referencing variables and functions with pm specification are shown in the following table.

		Reference Destination Object			
		Function with pmN Specification	Function with cmn Specification	Variable with pmN Specification	Variable with cmn Specification
Reference source object	Function with pmN specification	Can be called	Can be called	Can be referenced	Can be referenced
	Function with cmn specification	Cannot be called	Can be called	Can be referenced to only R0 relative variable	Can be referenced

- Variables with cmn specification need to be located in r0 relative sections. Therefore, if used in combination with the #pragma section directive, only the following attribute strings can be specified.
  r0_disp16, r0_disp23, r0_disp32, const, zconst, zconst23, default
  There are no limitations on the allocation section for functions with cmn specification.
  There are no limitations on the allocation section for variables with pmN specification or functions with pmN specification.
- When any of the -pic, -pirod, and -pid options is specified, defining a variable or function with cmn specification will cause an error.

The following shows specification examples.
These examples assume that the -Xmulti_level=1 is specified.

Example 1.

```
#pragma section r0_disp16
int     i;                      //.zbss.cmn
--------------------------------------------------------------------------------
#pragma pmodule pm2
int     i;                      //.bss.pm2
int     j = 5;                  //.data.pm2
const int       k = 10;         //.const.pm2
void func(void)                 //.text.pm2
{
        func2("abcde");         //"abcde" = .const.pm2
}
--------------------------------------------------------------------------------
#pragma pmodule pm2
#pragma section r0_disp16
int     i;                      //.zbss.pm2
--------------------------------------------------------------------------------
#pragma section r0_disp16
#pragma pmodule pm2
int     i;                      //.zbss.pm2
```

Example 2.

```
extern  int     i;
#pragma pmodule pm2
int     i;                      //.bss.pm2 (No warning)
```

Example 3.

```
#pragma pmodule pm2
extern  int     i;
int     i;                      //.bss.pm2 (No warning)
```

Example 4.

```
#pragma pmodule cmn
extern  int     i;
#pragma pmodule pm2
int     i;                      //.bss.cmn (No warning)
```

4.2.6.11 Specifying alignment value for branch destination addresses

The alignment value for function start addresses and branch destination addresses is set to 4.

```
#pragma align4 ( function-specification [, function-specification]... )Note
  function-specification: function-name [(spec)]
```

Note The outer parentheses can be omitted.

- One of the following can be specified as *spec*.

Spec	Description
function	Sets the alignment value for function start addresses to 4.
loop	Sets the alignment value for function start addresses and the start addresses of all loops to 4.
innermostloop	Sets the alignment value for function start addresses and the start address of the innermost loop to 4.
all	Sets the alignment value for function start addresses and all branch destination addresses to 4.

An error will occur if any other item is specified for spec.
If the specification of spec is omitted, it is assumed that function has been specified.

- If this option is specified simultaneously with the -Xalign4 option, the specification by #pragma align4 becomes valid.
- #pragma align4 can be specified only once for a single function. If it is specified more than once, an error will occur.
- If a function that does not specify #pragma align4 is included or an object module file or a standard library that has been generated through compilation without using the -Xalign4 option is specified for linkage, the warning W0561322 will be output at linkage but program execution will have no problem.

4.2.6.12 Detection of stack smashing [Professional Edition only]

The function for detecting stack smashing is implemented by the -Xstack_protector option, -Xstack_protector_all option, or the #pragma directives described in this section.

```
#pragma stack_protector ( function-specification [, function-specification]... )Note
  function-specification: function-name [(num=value)]
#pragma no_stack_protector ( function-name [, function-name]... )Note
```

Note The outer parentheses can be omitted.

- Generates a code for detection of stack smashing at the entry and end of a function. A code for detection of stack smashing indicates the instructions for executing the three processes shown below.
  (1) A 4-byte area is allocated just before the local variable area (in the direction towards address 0xFFFFFFFF) at the entry to a function, and the value specified by <number> is stored in the allocated area.
  (2) At the end of the function, whether the 4-byte area in which <number> was stored has been rewritten is checked.
  (3) If the value has been rewritten in (2), the __stack_chk_fail function is called as the stack has been smashed.
- A decimal number from 0 to 4294967295 should be specified in <number>. If the specification of <number> is omitted, the compiler automatically specifies the number.
- The __stack_chk_fail function needs to be defined by the user and the processing to be executed upon detection of stack smashing should be written.
  Note the following items when defining the __stack_chk_fail function.
  - The only possible type of return value is void and the __stack_chk_fail function does not have formal parameters.
  - Do not define the function as static.
  - It is prohibited to call the __stack_chk_fail function as a normal function.
  - The __stack_chk_fail function does not generate a code for detection of stack smashing regardless of the -Xstack_protector and -Xstack_protector_all options, and #pragma stack_protector.
  - Prevent returning to the caller, that is, the function where stack smashing was detected by taking measures such as calling abort() in the __stack_chk_fail function to terminate the program.
  - When this facility is used for a function for which PIC (see "8.6 PIC/PID Facility") is performed, PIC should also be performed for the __stack_chk_fail function.
- 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 -Xstack_protector option and -Xstack_protector_all option.
- If this option is used simultaneously with #pragma stack_protector, the -Xstack_protector option, or the -Xstack_protector_all option, the specification by #pragma becomes valid.
- #pragma stack_protector cannot be specified simultaneously with the following #pragma directives.
  #pragma inline_asm, #pragma inline, #pragma stack_protector, #pragma no_stack_protector
- #pragma no_stack_protector cannot be specified simultaneously with the following #pragma directives.
  #pragma stack_protector, #pragma no_stack_protector

Example

- <C source>

```
#include <stdio.h>
#include <stdlib.h>

#pragma stack_protector f1(num=1234)
void f1()   // Sample program in which the stack is smashed
{
    volatile char str[10];
    int i;
    for (i = 0; i <= 10; i++){
      str[i] = i; // Stack is smashed when i=10
    }
}

void __stack_chk_fail(void)
{
    printf("stack is broken!");
    abort();
}
```

- <Output code>

```
_f1:
        .stack _f1 = 16
        add 0xFFFFFFF0, r3
        movea 0x000004D2, r0, r1  ; The specified <number> 1234 is stored in the
                                  ; stack area.
        st.w r1, 0x0000000C[r3]
        mov 0x00000000, r2
        br9 .BB.LABEL.1_2
.BB.LABEL.1_1: ; bb
        movea 0x00000002, r3, r5
        add r2, r5
        st.b r2, 0x00000000[r5]
        add 0x00000001, r2
.BB.LABEL.1_2: ; bb7
        cmp 0x0000000B, r2
        blt9 .BB.LABEL.1_1
.BB.LABEL.1_3: ; return
        ld.w 0x0000000C[r3], r1   ; Data is loaded from the location where <number>
        movea 0x000004D2, r0, r12 ; was stored at the entry to a function and
        cmp r12, r1               ; it is compared with the specified num 1234.
        bnz9 .BB.LABEL.1_5        ; If they do not match, a branch occurs.
.BB.LABEL.1_4: ; return
        dispose 0x00000010, 0x00000000, [r31]
.BB.LABEL.1_5: ; return
        br9 ___stack_chk_fail     ; __stack_chk_fail is called.

___stack_chk_fail:
        .stack ___stack_chk_fail = 4
        prepare 0x00000001, 0x00000000
        mov #.STR.1, r6
        jarl _printf, r31
        jarl _abort, r31
        dispose 0x00000000, 0x00000001, [r31]
```

4.2.6.13 Half-precision floating-point type [Professional Edition only] [V1.05.00 or later]

The half-precision floating-point type can be used.
The half-precision floating-point type has the following features.

- The type name is __fp16.
- The size is two bytes and the alignment condition is also two bytes.
- The internal representation of data conforms to binary16 of IEEE754-2008.
  - The sign is one bit, the exponent is five bits, and the mantissa is 10 bits (11 bits when a hidden bit is included).
  - The bias of the exponent is 0xf. (Example: 1.0 is 0x3c00 in the hexadecimal notation.)
- The only supported operations are assignment between __fp16 type values, type conversion from __fp16 to float, and type conversion from float to __fp16. Other operations are to be performed after data has been converted into the float type, and the result will have the same type as that for when performing the same operation for the float type. Similarly, type conversion from __fp16 to double is to be performed after data has been converted into the float type.
- Denormal numbers are not supported for type conversion from float to __fp16, and they will be flushed to normal numbers in accordance with the rounding mode.
- Only -Xround=nearest is available as the rounding mode.
- There is no suffix for floating constants.
- This type cannot be specified as the parameter type or return type of a function. To pass a value of the __fp16 type between functions, pass it by casting it to another type (e.g. float), by using a pointer, or by using a structure argument that has a member of the __fp16 type.
- If the called function does not have a parameter type[Note], the value will be passed after the type is converted into float by default argument promotion and then further converted into double.

  Note This applies when there is no prototype declaration or parameter string, or there are a variable number of arguments.

- If this type is specified for an argument, the value is passed after being converted into the parameter type. If there is no parameter type, the value will be passed after the type is converted into float by default argument promotion and then converted into double.
- This type can be specified for a structure member, union member, or array element. This type cannot be specified for a bit field member.

Example

```
extern __fp16 hpvar1, hpvar2, hpvar3;
extern float fvar;
extern double dvar;
extern int ivar;

/* External variable definition */
__fp16 hpvar = 1.0;

void fun() {
  /* Constant assignment */
  hpvar = 1.0;

  /* Assignment to and from __fp16 */
  hpvar1 = hpvar2;

  /* Type conversion to single-precision floating-point number */
  fvar = hpvar;                                  /* Same as "fvar = (float)hpvar;" */

  /* Type conversion to double-precision floating-point number */
  dvar = hpvar;                            /* Same as "dvar = (double)(float)hpvar;" */

  /* Type conversion from double-precision floating-point number */
  hpvar = dvar;                            /* Same as "hpvar = (__fp16)(float)dvar;" */

  /* Type conversion to integer */
  ivar = hpvar;                               /* Same as "ivar = (int)(float)hpvar;" */

  /* Type conversion from integer */
  hpvar = ivar;                            /* Same as "hpvar = (__fp16)(float)ivar;" */

  /* Arithmetic operation */
  hpvar3 = hpvar1 + hpvar2;  /* Same as "hpvar3 = (__fp16)((float)hpvar1 +
(float)hpvar2;)" */
}
```

4.2.6.14 Detection of writing to control registers or insertion of synchronization processing [Professional Edition only] [V1.06.00 or later]

When multiple control registers are updated continuously by a store instruction of the RH850, the order in which the control registers are updated may not match the order in which they were written to the source file. Synchronization processing needs to be inserted in order to control the order in which the control registers are updated.

CC-RH is capable of detecting writing to control registers and displaying information on the writing process, as well as inserting synchronization processing of a fixed form.

Code indicating access to control registers will be detected when all of the following conditions are satisfied.

- An expression shall cast a single integer constant to a pointer to a volatile qualifier and indirectly reference it with the unary * operator or -> operator.

```
*(volatile int*)0xffff0000 = x;
((volatile struct ST*)0xffff0004)->member = y;
```

  In this case, the integer constant indicates the address of the control register.
  An expression including a variable instead of a constant integer or an expression including multiple casts may not be detected.

- The above integer constant shall be within the range specified by #pragma register_group.

Specify the address range and group information of control registers with #pragma register_group.
Specify #pragma register_group in the following format.

```
#pragma register_group start-address, end-address[, id="group-ID"]
```

- Specify the start address and end address with unsigned integers. An octal, decimal, or hexadecimal integer can be used.
  The end address is treated as its address belongs to the group.

  Example Specify a 16-byte area starting from address 0x100 as follows:

```
#pragma register_group 0x100, 0x10f
```

- The group-ID is an identifier for specifying the group to which the control register belongs.
  The usable characters for the group-ID are only alphabetic characters (a-z or A-Z; case-sensitive), numbers (0-9), and underscore (_).
  The length of the group-ID is not limited.
- To specify the same group which is not continuous in the address space, the same group-ID can be specified in more than one #pragma register_group directive.
- The group-ID can be omitted. If omitted, writing to that area is treated as an operation that is not continuous with any other write operations.
- An error will be output in any of the following cases.
  - When the start address is larger than the end address
  - When an unusable character for the group-ID is used
  - When the address ranges specified in multiple #pragma register_group directives overlap with each other

How to detect writing to control registers and insert synchronization processing is described below, based on the following example of input source code.

Example of input source code

```
#pragma register_group 0xfedf0000, 0xfedfffff, id="CPU"
#pragma register_group 0xfee00000, 0xfee0ffff, id="0"

#define REG1 (*(volatile unsigned char*)0xfedf0000)    /* Control register of group
"CPU" */
#define REG2 (*(volatile unsigned char*)0xfedf0001)    /* Control register of group
"CPU" */
#define REG_Z (*(volatile unsigned short*)0xfee00000)  /* Control register of group
"0" */

void func(void) {
   REG1 = 0;
   REG2 = 1;
   REG_Z = 2;
}
```

(a) How to detect writing to a control register

If the above example is compiled with the -store_reg=list option specified, write operations are determined as shown below and a message indicating that writing was performed is output to the standard error output.

- Since REG1 and REG2 belong to the same group, synchronization processing does not need to be applied to writing to REG1.
- Since REG2 and REG_Z belong to different groups, synchronization processing needs to be applied to writing to REG2.
- Since whether there is writing to the same group after writing to REG_Z is not clear, synchronization processing needs to be applied to writing to REG_Z.

```
src.c(10):M0536001:control register is written.(id=CPU, 0xfedf0001)
src.c(11):M0536001:control register is written.(id=0, 0xfee00000)
```

Note that synchronization processing is determined to be necessary even in cases where writing to a control register is followed by a function call or access to an unknown address in memory that cannot be detected as writing to a control register.

(b) How to detect all writings to control registers

If the same example is compiled with the -store_reg=list_all option specified, a message is output for writing to all of the control registers specified by #pragma register_group without determining whether continuous writing to the same group is performed.

```
src.c(9):M0536001:control register is written.(id=CPU, 0xfedf0000)
src.c(10):M0536001:control register is written.(id=CPU, 0xfedf0001)
src.c(11):M0536001:control register is written.(id=0, 0xfee00000)
```

(c) How to insert synchronization processing after writing to a control register

If the same example is compiled with the -store_reg=sync option specified, write operations are determined in the same manner as (a) and synchronization processing is inserted in the output code instead of a message being output. In synchronization processing, a combination of loading from the same control register and syncp instruction is output.

Output example

```
_func:
        .stack _func = 0
        movhi 0x0000FEDF, r0, r2
        st.b r0, 0x00000000[r2]
         ; Synchronization processing is not inserted because writing to the
         ; same group occurs later.
        movhi 0x0000FEDF, r0, r2
        mov 0x00000001, r5
        st.b r5, 0x00000001[r2]
        ld.bu 0x00000001[r2], r10  ; Synchronization processing is inserted.
        syncp                      ;

        movhi 0x0000FEE0, r0, r2
        mov 0x00000002, r5
        st.h r5, 0x00000000[r2]
        ld.hu 0x00000002[r2], r10  ; Synchronization processing is inserted.
        syncp                      ;
        jmp [r31]
```

[Caution]

- When #pragma register_group is used with the -Xmerge_file option at the same time, even though a #pragma register_group directive that is incompatible between source files is specified, an error will not occur and the result may be unintended. It is recommended to include #pragma register_group in the include file and share it between source files.
- If there is a possibility that an exception will occur between writing to a control register and the synchronization processing, manually write the synchronization processing in the exception handler as necessary.
- Successive writing to control registers belonging to the same group may not be detected in certain circumstances. In such a case, CC-RH may output incorrect detection messages or insert unnecessary synchronization processing.

4.2.7 Modification of C source

By using expanded function object with high efficiency can be created. However, as expanded function is adapted in RH850 family, C source needs to be modified so as to use in other than RH850 family.

Here, 2 methods are described for shifting to the CC-RH from other C compiler and shifting to C compiler from the CC-RH.

<From other C compiler to the CC-RH>

- #pragma[Note]
  C source needs to be modified, when C compiler supports the #pragma. Modification methods are examined according to the C compiler specifications.
- Expanded Specifications
  It should be modified when other C compilers are expanding the specifications such as adding keywords etc. Modified methods are examined according to the C compiler specifications.

Note #pragma is one of the pre-processing directives supported by C90 and C99. The character string next to #pragma is made to be recognized as directives to C compiler. If that directive does not supported by the compiler, #pragma directive is ignored and the compiler continues the process and ends normally.

<From the CC-RH to other C compiler>

- The CC-RH, either deletes key word or divides # fdef in order shift to other C compiler as key word has been added as expanded function.

Example 1. Disable the keywords

```
#if !defined(__CCRH__)
#define inline          /* considered inline function as normal function */
#endif
```

Example 2. Change to other type

```
#if !defined(__CCRH__)
#define _Bool char      /* change _Bool type variable to char type variable */
#endif
```

5. ASSEMBLY LANGUAGE SPECIFICATIONS

This chapter explains the assembly language specifications supported by the CC-RH assembler.

5.1 Description of Source

This section explains description of source, expression, and operators.

5.1.1 Description

An assembly language statement consists of a "symbol", a "mnemonic", "operands", and a "comment".

```
[symbol][:]       [mnemonic]     [operand], [operand]    ;[comment]
```

Separate labels by colons or one or more whitespace characters. Whether colons or spaces are used, however, depends on the instruction coded by the mnemonic.

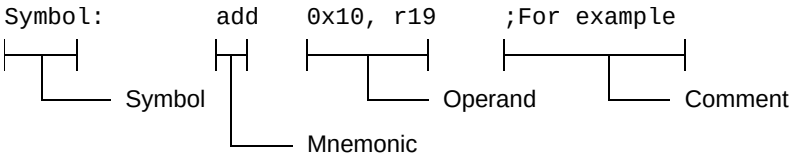
It is irrelevant whether blanks are inserted in the following location.

- Between the symbol name and colon
- Between the colon and mnemonic
- Before the second and subsequent operands
- Before semicolon that indicates the beginning of a comment

One or more blank is necessary in the following location.

- Between the mnemonic and the operand

Figure 5.1 Organization of Assembly Language Statement



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 asembler 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.

Table 5.1 Language Characters and Usage of Characters

Character	Usage
Numerals	Constitutes an identifier and constant
Lowercase letter (a-z)	Constitutes a mnemonic, identifier, and constant
Uppercase letter (A-Z)	Constitutes a mnemonic, identifier, and constant
@	Constitutes an identifier

Character	Usage
_ (underscore)	Constitutes an identifier
.(period)	Constitutes an identifier and constant
, (comma)	Delimits an operand
: (colon)	Delimits a label
; (semicolon)	Beginning of comment
*	Multiplication operator
/	Division operator
+	Positive sign and addition operator
- (hyphen)	Negative sign and subtraction operator
' (single quotation)	Character constant
<	Relational operator
>	Relational operator
( )	Specifies an operation sequence
$	Symbol indicating the start of a control instruction equivalent to an assembler option Symbol specifying relative addressing gp offset reference of label Constitutes an identifier
=	Relational operator
!	Beginning immediate addressing and negation operator
Δ (blank)	Field delimiter
~	Concatenation symbol (in macro body)
&	Logical product operator
#	References the absolute address of a label and begins a comment (when used at the beginning of a line)
[ ]	Indirect indication symbol
"(double quotation)	Start and end of character string constant
%	ep offset reference of a label and remainder operator
<<	Left shift operator
>>	Right shift operator
|	Logical sum operator
^	Exclusive OR operator

(b) Character data

Character data refers to characters used to write character string constant, character constant, and the quote-enclosed operands of some control instructions.

Caution Character data can use all characters (including multibyte kanji, although the encoding depends on the OS).

(c) Comment characters

Comment characters are used to write comments.

Caution Comment characters and character data have the same character set.

(2) 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 can be classified as shown below, depending on their purpose and how they are defined.

Symbol Type	Purpose	Definition Method
Name	Used as names for addresses and data objects in source programs.	Write in the symbol field of a Symbol definition directive.
Label	Used as labels for addresses and data objects in source programs.	Write a symbol followed by a colon (:).
External reference name	Used to reference symbols defined by other source modules.	Write in the operand field of an external reference directive.
Section name	Used at link time.	Write in the symbol field of a section definition directive.
Macro name	Use to name macros in source programs.	Write in the symbol field of macro directive.

(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).
- 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 "5.5 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.
- The assembler distinguishes between lowercase and uppercase characters.
- When a label is written in a symbol field, the colon (:) must appear immediately after the label name.

Example Correct symbols

```
CODE01  .cseg               ; "CODE01" is a section name.
VAR01   .set    0x10        ; "VAR01" is a name.
LAB01:  .dw     0           ; "LAB01" is a label.
```

Example Incorrect symbols

```
1ABC    .set    0x3         ; The first character is a digit.s
LAB     mov     1, r10      ; "LAB"is a label and must be separated from the mne-
monic
                            ; field by a colon ( : ).
FLAG:   .set    0x10        ; The colon ( : ) is not needed for symbols.
```

Example A statement composed of a symbol only

```
ABCD:                       ; ABCD is defined as a label.
```

(3) 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.

Example Correct mnemonics

```
mov     1, r10
```

Example Incorrect mnemonics

```
mov1, r10       ; There is no blank between the mnemonic and operand fields.
mo v    1, r10  ; The mnemonic field contains a blank.
MOVE           ; This is an instruction that cannot be coded in the mnemonic field.
```

(4) 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 (,).
The following types of data can appear in the operand field:

- Constants (numeric constants, character constants, character string constants)
- Register names
- Symbols
- Expressions

See the user's manual of the target device for the format and notational conventions of instruction set operands.
The following sections explain the types of data that can appear in the operand field.

(a) Constants

A constant is a fixed value or data item and is also referred to as immediate data.
There are numeric constants, character constants and character string constants.

<1> 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 any message).

Type	Notation	Example
Binary	Append an "0b" or "0B" suffix to the number.	0b1101
Octal	Append an "0" suffix to the number.	074
Decimal	Simply write the number.	128
Hexadecimal	Append an "0x" or "0X" suffix to the number.	0xA6

Floating constants consist of the following elements. Specify the exponent and mantissa as decimal constants. Do not use (3), (4), or (5) if an exponent expression cannot be used.

(1) sign of mantissa part ("+" is optional)
(2) mantissa part
(3) 'e' or 'E' indicating the exponent part
(4) sign of exponent part ("+" is optional)
(5) exponent part

Example

```
123.4
-100.
10e-2
-100.2E+5
```

You can indicate that the number is a floating constant by appending "0f" or "0F" to the front of the mantissa.

Example

```
0f10
```

<2> 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[Note].

If any of the escape sequences listed below is specified in " ' " and " ' ", the assembler regards the sequence as being a single character.

Example

```
'A'             ; 0x00000041
' '             ; 0x00000020 (1 blank)
```

Note If a character constant is specified, the assembler assumes that an integer having the value of that character constant is specified.

Table 5.2 Value and Meaning of Escape Sequence

Escape Sequence	Value	Meaning
\0	0x00	null character
\a	0x07	Alert
\b	0x08	Backspace
\f	0x0C	Form feed
\n	0x0A	Line feed
\r	0x0D	Carriage return
\t	0x09	Horizontal tab
\v	0x0B	Vertical tab
\\	0x5C	Back slash
\'	0x27	Single quotation marks
\"	0x22	Double quotation mark
\?	0x3F	Question mark
\ddd	0 to 0377	Octal number of up to 3 digits (0 <= d <= 7) [Note]
\xhh	0 to 0xFF	Hexadecimal number of up to 2 digits (0 <= h <= 9, a <= h <= f, or A <= h <= F)

Note If a value exceeding "\377" is sp value of the escape sequence becomes the lower 1 byte. Cannot be of value more than 0377. For example value of"\777"is 0377.

<3> Character string constants

A character-string constant is expressed by enclosing a string of characters from those shown in "(1) Character set", in a pair of single quotation marks (").

Example

```
"ab"            ; 0x6162
"A"             ; 0x41
" "             ; 0x20 (1 blank)
```

(b) Register names

The following registers can be named in the operand field:

- r0, zero, r1, r2, hp, r3, sp, r4, gp, r5, tp, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, ep, r31, lp

r0 and zero (Zero register), r2 and hp (Handler stack pointer), r3 and sp (Stack pointer), r4 and gp (Global pointer), r5 and tp (Text pointer), r30 and ep (Element pointer), r31 and lp (Link pointer) shows the same register.

Remark For the ldsr and stsr instructions, the PSW, and system registers are specified by using the numbers. Further, in assembler, PC cannot be specified as an operand.

(c) Symbols

The assembler supports the use of symbols as the constituents of the absolute expressions or relative expressions that can be used to specify the operands of instructions and directives.

(d) Expressions

An expression is a combination of constants and symbols, by an operator.
Expressions can be specified as instruction operands wherever a numeric value can be specified.
See "5.1.2 Expressions and operators" for more information about expressions.

Example

```
TEN     .set    0x10
        mov     TEN - 0x05, r10
```

In this example, "TEN - 0x05" is an expression.
In this expression, a symbol and a numeric value are connected by the - (minus) operator. The value of the expression is 0x0B, so this expression could be rewritten as "mov 0x0B, r10".

(5) Comment

Describe comments in the comment field, after a semicolon (;).
The comment field continues from the 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".

5.1.2 Expressions and operators

An expression is a symbol, constant, an operator combined with one of the above, or a combination of operators.
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 assembler supports the operators shown in "Table 5.3 Operator Types". Operators have priority levels, which determine when they are applied in the calculation. The priority order is shown in "Table 5.4 Operator Precedence Levels".
The order of calculation can be changed by enclosing terms and operators in parentheses "( )".

Example

```
mov32   5 * (SYM + 1), r12
```

In the above example, "5 * (SYM+1)" is an expression. "5" is the 1st term, "SYM" is the 2nd term, and "1" is the 3rd term. The operators are "*", "+", and "( )".

Table 5.3 Operator Types

Operator Type	Operators
Arithmetic operators	+, -, *, /, %, +sign, -sign
Logic operators	!, &, |, ^
Relational operators	==, !=, >, >=, <, <=, &&, ||
Shift operators	>>, <<

Operator Type	Operators
Byte separation operators	HIGH, LOW
2-byte separation operators	HIGHW, LOWW, HIGHW1
Section operators	STARTOF, SIZEOF
Other operator	( )

The above operators can also be divided into unary operators and binary operators.

Unary operators	+sign, -sign, !, HIGH, LOW, HIGHW, LOWW, HIGHW1
Binary operators	+, -, *, /, %, &, |, ^, ==, =, >, >=, <, <=, >>, <<, &&, ||

Table 5.4 Operator Precedence Levels

Priority	Level	Operators
Higher	1	+sign, -sign, !, HIGH,LOW,HIGHW,HIGHW1,LOWW,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 unsigned 32-bit values. However, terms of expressions in multiplication, division, and modulo operation and the second term of logical shift are handled as signed 32-bit values.
- If the value of a constant exceeds 32 bits, the overflow value is ignored.
- In division, the decimal fraction part is discarded.
  If the divisor is 0, an error occurs.
- Negative values are represented as two's complement.
- Relative expressions are evaluated as 0 at the time when the source is assembled (the evaluation value is determined at link time).

Table 5.5 Evaluation examples

Expression	Evaluation
2 + 4 * 5	22
(2 + 3) * 4	20
10/4	2
0 - 1	0xFFFFFFFF
EXT[Note] + 1	0

Note EXT: External reference symbols

5.1.3 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]

```
        .org    0x100
START:  jr      START + 6       ; (1)
```

(1) The jr 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]

```
        .org    0x100
BACK:   jr      BACK - 6        ; (1)
```

(1) The jr instruction causes a jump to "address assigned to BACK minus 6", namely, to address "0x100 - 0x6 = 0xFA" when BACK label is 0x100.

```
*
```

Multiplacation 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]

```
TEN     .set    0x10
        mov     TEN * 3, r10    ; (1)
```

(1) With the .set directive, the value "0x10" is defined in the symbol "TEN".
The expression "TEN * 3" is the same as "0x10 * 3" and returns the value "0x30".
Therefore, (1) in the above expression can also be described as: mov 0x30, r10.

```
/
```

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     256 / 50, r10           ; (1)
```

(1) The result of the division "256 / 50" is 5 with remainder 6.
The operator returns the value "5" that is the integer part of the result of the division.
Therefore, (1) in the above expression can also be described as: mov 5, r10.

%

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     256 % 50, r10   ; (1)
```

(2) The result of the division "256 / 50" is 5 with remainder 6.
The MOD operator returns the remainder 6.
Therefore, (1) in the above expression can also be described as: mov 6, r10.

+sign

Returns the value of the term as it is.

[Function]

Returns the value of the term of an expression without change.

[Application example]

```
FIVE    .set    +5      ; (1)
```

(1) The value "5" of the term is returned without change.
The value "5" is defined in symbol "FIVE" with the .set directive.

-sign

The term value 2 complement is sought.

[Function]

Returns the value of the term of an expression by the two's complement.

[Application example]

```
NO      .set    -1      ; (1)
```

(1) -1 becomes the two's complement of 1.
0000 0000 0000 0000 0000 0000 0000 0001 becomes:
1111 1111 1111 1111 1111 1111 1111 1111
Therefore, with the .set directive, the value "0xFFFFFFFF" is defined in the symbol "NO".

5.1.4 Logic operators

The following 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     !0x3, r10       ; (1)
```

(1) Logical negation is performed on "0x3" as follows:
0xFFFFFFFC is returned.

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     0x6FA & 0xF, r10        ; (1)
```

(1) AND operation is performed between the two values "0x6FA" and "0xF" as follows:
The result "0xA" is returned. Therefore, (1) in the above expression can also be described as:
mov 0xA, r10.

	0000	0000	0000	0000	0000	0110	1111	1010
&)	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     0xA | 0b1101, r10       ; (1)
```

(1) OR operation is performed between the two values "0xA" and "0b1101" as follows:
The result "0xF" is returned.
Therefore, (1) in the above expression can also be described as: mov 0xF, r10.

	0000	0000	0000	0000	0000	0000	0000	1010
|)	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]

```
mov32   0x9A ^ 0x9D, r12        ; (1)
```

(1) XOR operation is performed between the two values "0x9A" and "0x9D" as follows:
The result "0x7" is returned.
Therefore, (1) in the above expression can also be described as: mov32 0x7, r12.

	0000	0000	0000	0000	0000	0000	1001	1010
^)	0000	0000	0000	0000	0000	0000	1001	1101
	0000	0000	0000	0000	0000	0000	0000	0111

5.1.5 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.
&&	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.

==

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.

&&

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.

5.1.6 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.

The sign bit is not shifted.

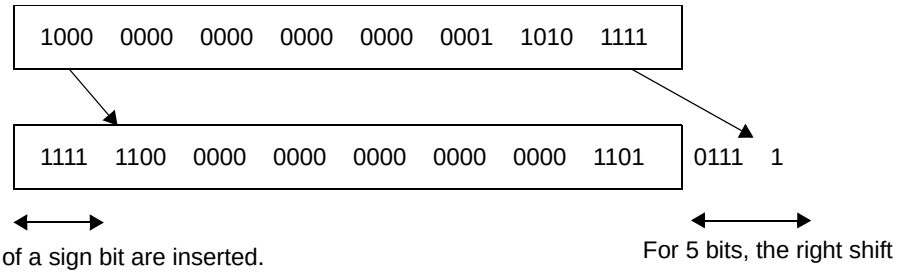
The sign bit is inserted in the high-order bits, the same number of times as the number of bits that were shifted.

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]

```
mov32   0x800001AF >> 5, r20     ; (1)
```

(1) The value "0x800001AF" is shifted 5 bits to the right, leaving the sign bit.
"0xFC00000D" is forwarded to r20.
Therefore, (1) in the above example can also be described as: mov32 0xFC00000D,r20.



```
<<
```

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.

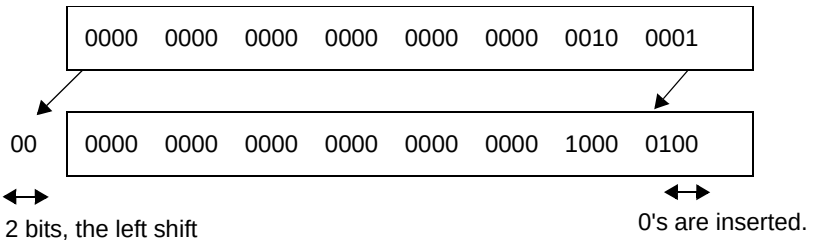
Zeros equivalent to the specified number of bits shifted move into the low-order bits.

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]

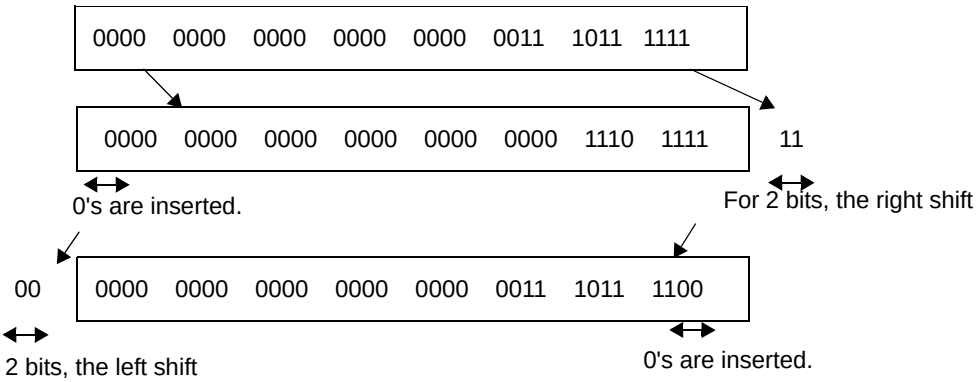
```
mov32   0x21 << 2, r20          ; (1)
```

(1) This operator shifts the value "0x21" to the left by 2 bits.
"0x84" is forwarded to r20.
Therefore, (1) in the above example can also be described as: mov32 0x84, r20.




```
mov32   0x3BF >> 2 << 2, r20    ; (2)
```

(2) This operator shifts the value "0x3B" to the right by 2 bits, and shifts to the left by 2 bits.
"0x3BC" is forwarded to r20.
Therefore, (2) in the above example can also be described as: mov32 0x3BC, r20.



5.1.7 Byte separation operators

The following byte separation operators are available.

Operator	Overview
HIGH	Returns the high-order 8-bit value of a term.
LOW	Returns the low-order 8-bit value of a term.

HIGH

Returns the high-order 8-bit value of a term.

[Function]

Returns the high-order 8-bit value of a term.

[Application example]

```
mov     HIGH(0xC08), r10        ; (1)
```

(1) By executing a mov instruction, this operator returns the high-order 8-bit value "0xC" of the expression "0xC08". Therefore, (1) in the above example can also be described as: mov 0xC, r10.

LOW

Returns the low-order 8-bit value of a term.

[Function]

Returns the low-order 8-bit value of a term.

[Application example]

```
mov     LOW(0xC08), r10         ; (1)
```

(1) By executing a mov instruction, this operator returns the low-order 8-bit value "0x8" of the expression "0xC08". Therefore, (1) in the above example can also be described as: mov 0x8, r10.

5.1.8 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.
HIGHW1	The value calculated by adding the value at the 15th bit to the uppermost 16 bits of the term.

HIGHW

Returns the high-order 16-bit value of a term.

[Function]

Returns the high-order 16-bit value of a term.

[Application example]

```
movea   HIGHW(0x12345678), R0, r10      ; (1)
```

(1) By executing a movea instruction, this operator returns the high-order 16-bit value "0x1234" of the expression "0x12345678".
Therefore, (1) in the above example can also be described as: movea 0x1234, R0, r10.

LOWW

Returns the low-order 16-bit value of a term.

[Function]

Returns the low-order 16-bit value of a term.

[Application example]

```
movea   LOWW(0x12345678), R0, r10       ; (1)
```

(1) By executing a movea instruction, this operator returns the low-order 16-bit value "0x5678" of the expression "0x12345678".
Therefore, (1) in the above example can also be described as: movea 0x5678, R0, r10.

HIGHW1

The value calculated by adding the value at the 15th bit to the uppermost 16 bits of the term.

[Function]

The value calculated by adding the value at the 15th bit to the uppermost 16 bits of the term.
If the value of the upper 16 bits is 0xffff while the value of the 15th bit is 1, HIGHW1 returns the value of 0.

[Application example]

```
movhi   HIGHW1(0x12348765), R0, r10     ; (1)
```

(1) Given the value 0x12348765, a movhi instruction adds the value at the 15th bit (1) to the top 16 bits (0x1234), returning the value 0x1235.
Therefore, (1) in the above example can also be described as: movhi 0x1235, R0, r10.
If the value of the upper 16 bits is 0xffff while the value of the 15th bit is 1, HIGHW1 returns the value of 0.

5.1.9 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 term section after linking.

[Function]

Returns the start address of the term section after linking.

[Application example]

```
.DW     STARTOF(.text)          ; (1)
```

(1) Allocates a 4-byte area, and initializes it with the start address of the .text section.

To use this operator in conjunction with SIZEOF:

```
.DW         STARTOF(.data) + SIZEOF(.data)
```

[Caution]

- STARTOF can only be written as an operand of the data definition directive, .dw.
- 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.

SIZEOF

Returns the size of the term section after linking.

[Function]

Returns the size of the term section after linking.

[Application example]

```
.DW     SIZEOF(.text)           ; (1)
```

(1) Allocates a 4-byte area, and initializes it with the size of the .text section.

To use this operator in conjunction with STARTOF:

```
.DW         STARTOF(.data) + SIZEOF(.data)
```

[Caution]

- SIZEOF can only be written as an operand of the data definition directive, .dw.
- This operator can be specified in combination with STARTOF 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.

5.1.10 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     (4 + 3) * 2, r10
```

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 5.4 Operator Precedence Levels", for the order of precedence of operators.

5.1.11 Restrictions on operations

An expression consists of a "constant", "symbol", "label reference", "operator", and "parentheses".It indicates a value consisting of these elements. The expression distinguishes between Absolute expression and Relative expressions.

(1) 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.

(a) Constant expression

If a reference to a previously defined symbol is specified, assumes 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.

Example

```
SYM1    .set    0x100       --Define symbol SYM1
        mov     SYM1, r10   --SYM1, already defined, is treated as a constant
                              expression.
```

(b) 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 reference to a symbol defined as a constant with a symbol definition directive in the same file, although undefined at that point. If a reference to a previously defined symbol is specified, assumes that the "constant" of the value defined for the symbol has been specified.

Example

```
        add     SYM1 + 0x100, r11   --SYM1 is an undefined symbol at this point
SYM1    .set    0x10                --Defines SYM1
```

(c) Label reference

The following expressions are related to label reference ("±" is either "+" or "-").

- Label reference - label reference
- Label reference - label reference ± constant expression

Here is an example of an expression related to a label reference.

Example

```
mov     $label1 - $label2, r11
```

A "reference to two labels" as shown in this example must be referenced as follows.

- The same section has a definition in the specified file.
- Same reference method (such as $label and $label, and #label and #label)

When not meeting these conditions, a message is output, and assembly is canceled.

The .DW directive can be assembled if two label accesses are absolute address references, even if the definitions are in different sections of different files.

(2) 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 label reference. The following format[Note 2] is treated as an relative expression.

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. The absolute value system and the relative value system 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.

(a) Label reference
The following expressions are related to label reference ("±" is either "+" or "-").

- Label reference
- Label reference + constant expression
- Label reference - symbol
- Label reference - symbol ± constant expression

Here is an example of an expression related to a label reference.

Example

```
SIZE    .set    0x10
        add     #label1, r10
        add     #label1 + 0x10, r10
        add     #label2 - SIZE, r10
        add     #label2 - SIZE + 0x10, r10
```

5.1.12 Identifiers

An identifier is a name used for symbols, labels, macros etc.
Identifiers are described according to the following basic rules.

- Identifiers consist of alphanumeric characters and symbols that are used as characters (@,_, ., $)
  However, the first character cannot be a number (0 to 9) or $.
- Reserved words cannot be used as identifiers.
  With regard to reserved words, see "5.5 Reserved Words".
- The assembler distinguishes between uppercase and lowercase.

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 5.6 List of Directives

Type	Directives
Section definition directives	.cseg, .dseg, .section, .org
Symbol definition directives	.set, .equ
Compiler output directives	.file, .line, .stack, ._line_top, ._line_end
Data definition/Area reservation directives	.db, .db2/.dhw, .dshw, .db4/.dw, .db8/.ddw, .float, .double, .ds, .align
External definition/External reference directives	.public, .extern
Macro directives	.macro, .local, .rept, .irp, .exitm, .exitma, .endm

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 is a block of routines or data of the same type. A "section definition directive" is a directive that declares the start or end of a section.

Sections are the unit of allocation in the optimizing linker.

Example

```
.cseg
  :
.dseg
  :
```

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.

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 the sections are broken into separate source program files, then they will be processed by the optimizing linker.

Section names cannot be referenced as symbols.

The following section definition directives are available.

Table 5.7 Section Definition Directives

Directive	Overview
.cseg	Indicates to the assembler the starting of a code section
.dseg	Indicates to the assembler the start of a data section
.section	Indicates to the assembler the start of a 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

.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.

[Use]

- The .cseg directive is used to describe instructions, .db, .dw directives, etc. in the code section defined by the .cseg directive.
- Description of one functional unit such as a subroutine should be defined as a single code section.

[Description]

- The start address of a code section can be specified with the .org directive.
- A relocation attribute defines a range of location addresses for a code section.

Table 5.8 Relocation Attributes of .cseg

Relocation Attribute	Explanation	Default Section Name	Default Value of Alignment Condition
TEXT	Allocates the program.	.text	2
PCTEXT [V1.07.00 or later]	Allocates position-independent programs.	.pctext	2
ZCONST	This section is for constant (read-only) data. It allocates a memory range (up to 32 Kbytes, in the positive direction from r0), referenced with 1 instruction using r0 and 16-bit displacement.	.zconst	4
ZCONST23	This section is for constant (read-only) data. It allocates a memory range (up to 4 Mbytes, in the positive direction from r0), referenced with 1 instruction using r0 and 23-bit displacement.	.zconst23	4
CONST	This section is for constant (read-only) data. It allocates a memory range (up to 4 Gbytes, in the positive direction from r0), referenced with 2 instructions using r0 and 32-bit displacement.	.const	4
PCCONST16 [V1.07.00 or later]	This section is for position-independent constant data (read-only). It allocates a memory range (up to ±32 Kbytes from __pc_data) referenced with 1 instruction using 16-bit displacement from the __pc_data symbol.	.pcconst16	4

Relocation Attribute	Explanation	Default Section Name	Default Value of Alignment Condition
PCCONST23 [V1.07.00 or later]	This section is for position-independent constant data (read-only). It allocates a memory range (up to ±4 Mbytes from __pc_data) referenced with 1 instruction using 23-bit displacement from the __pc_data symbol.	.pcconst23	4
PCCONST32 [V1.07.00 or later]	This section is for position-independent constant data (read-only). It allocates a memory range referenced with 2 instructions using 32-bit displacement from the __pc_data symbol.	.pcconst32	4

- An error will occur in any of the following cases.
  - When a relocation attribute other than those in "Table 5.8 Relocation Attributes of .cseg" is specified
  - If TEXT is specified when the -pic option has been specified
  - If PCTEXT is specified when the -pic option has not been specified
  - If CONST, ZCONST, or ZCONST23 is specified when the -pirod option has been specified
  - If PCCONST16, PCCONST23, or PCCONST32 is specified when the -pirod option has not been specified
- The allocation section of instructions or data when no symbol definition directive has been written is the ".text" section if the -pic option has not been specified and the ".pctext" section if the -pic option has been specified.
- By describing a section name in the symbol field of the .cseg directive, the code section can be named. If no section name is specified for a code section, the assembler will automatically give a default section name to the code section. The default section names of the code sections are shown below.
- The default section names have the relocation attributes shown above. Giving them any other attributes is not possible.

Example

```
test        .cseg   text
            nop
            nop
```

- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

.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.

[Use]

- The .ds directive is mainly described in the data section defined by the .dseg directive.

[Description]

- The start address of a data section can be specified with the .org directive.
- A relocation attribute defines a range of location addresses for a data section.
  The relocation attributes available for data sections are shown below.

Table 5.9 Relocation Attributes of .dseg

Relocation Attribute	Explanation	Default Section Name	Default Value of Alignment Condition
SDATA	Allocates a memory range (up to 64 Kbytes, combined with SBSS section), referenced with 1 instruction using gp and 16-bit displacement, having an initial value.	.sdata	4
SBSS	Allocates a memory range (up to 64 Kbytes, combined with SDATA section), referenced with 1 instruction using gp and 16-bit displacement, not having an initial value.	.sbss	4
SDATA23	Allocates a memory range (up to 8MKbytes, combined with SBSS23 section), referenced with 1 instruction using gp and 23-bit displacement, having an initial value.	.sdata23	4
SBSS23	Allocates a memory range (up to 8Mbytes, combined with SDATA23 section), referenced with 1 instruction using gp and 23-bit displacement, not having an initial value.	.sbss23	4
SDATA32 [V1.07.00 or later]	Allocates a memory range, referenced with 2 instructions using gp and 32-bit displacement, having an initial value.	.sdata32	4
SBSS32 [V1.07.00 or later]	Allocates a memory range, referenced with 2 instructions using gp and 32-bit displacement, not having an initial value.	.sbss32	4
TDATA	Allocates a memory range (up to 256 bytes, in the positive direction from ep), referenced with 1 instruction using ep, having an initial value.	.tdata	4

Relocation Attribute	Explanation	Default Section Name	Default Value of Alignment Condition
TDATA4	Allocates a memory range (up to 16 bytes, in the positive direction from ep), referenced with 1 instruction1 instruction using ep and 4-bit displacement, having an initial value.	.tdata4	4
TBSS4	Allocates a memory range (up to 16 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 4-bit displacement, not having an initial value.	.tbss4	4
TDATA5	Allocates a memory range (up to 32 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 5-bit displacement, having an initial value.	.tdata5	4
TBSS5	Allocates a memory range (up to 32 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 5-bit displacement, not having an initial value.	.tbss5	4
TDATA7	Allocates a memory range (up to 128 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 7-bit displacement, having an initial value.	.tdata7	4
TBSS7	Allocates a memory range (up to 128 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 7-bit displacement, not having an initial value.	.tbss7	4
TDATA8	Allocates a memory range (up to 256 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 8-bit displacement, having an initial value.	.tdata8	4
TBSS8	Allocates a memory range (up to 256 bytes, in the positive direction from ep), referenced with 1 instruction using ep and 8-bit displacement, not having an initial value.	.tbss8	4
EDATA	Allocates a memory range (up to 64 Kbytes, combined with EBSS section), referenced with 1 instruction using ep and 16-bit displacement, having an initial value.	.edata	4
EBSS	Allocates a memory range (up to 64 Kbytes, combined with EDATA section), referenced with 1 instruction using ep and 16-bit displacement, not having an initial value.	.ebss	4
EDATA23	Allocates a memory range (up to 8 Mbytes, combined with EBSS23 section), referenced with 1 instruction using ep and 16-bit displacement, having an initial value.	.edata23	4
EBSS23	Allocates a memory range (up to 8 Mbytes, combined with EDATA23 section), referenced with 1 instruction using ep and 16-bit displacement, not having an initial value.	.ebss23	4
EDATA32 [V1.07.00 or later]	Allocates a memory range, referenced with 2 instructions using gp and 32-bit displacement, having an initial value.	.edata32	4
EBSS32 [V1.07.00 or later]	Allocates a memory range, referenced with 2 instructions using gp and 32-bit displacement, not having an initial value.	.ebss32	4
ZDATA	Allocates a memory range (up to 32 Kbytes, combined with ZBSS section, in the negative direction from r0), referenced with 1 instruction using r0 and 16-bit displacement, having an initial value.	.zdata	4

Relocation Attribute	Explanation	Default Section Name	Default Value of Alignment Condition
ZBSS	Allocates a memory range (up to 32 Kbytes, combined with ZDATA section, in the negative direction from r0), referenced with 1 instruction using r0 and 16-bit displacement, not having an initial value.	.zbss	4
ZDATA23	Allocates a memory range (up to 4 Mbytes, combined with ZBSS23 section, in the negative direction from r0), referenced with 1 instruction using r0 and 23-bit displacement, having an initial value.	.zdata23	4
ZBSS23	Allocates a memory range (up to 4 Mbytes, combined with ZDATA23 section, in the negative direction from r0), referenced with 1 instruction using r0 and 23-bit displacement, not having an initial value.	.zbss23	4
DATA	Allocates a memory range (up to 4 Gbytes, combined with BSS section, in the negative direction from r0), referenced with 2 instructions using r0 and 32-bit displacement, having an initial value.	.data	4
BSS	Allocates a memory range (up to 4 Gbytes, combined with DATA section, in the negative direction from r0), referenced with 2 instructions using r0 and 32-bit displacement, not having an initial value.	.bss	4

Note If a section with the TDATA relocation attribute is defined in multiple files of source code, linkage of the code will lead to an error.

Note A specifiable section name is only a default section name in a section with the TDATA relocation attributes.

- An error will occur in any of the following cases.
  - When a relocation attribute other than those in "Table 5.9 Relocation Attributes of .dseg" is specified
  - When a machine language instruction or data definition directive is described in a section with BSS relocation
  - If SDATA32, SBSS32, EDATA32, or EBSS32 is specified when the -pid option has not been specified
  - If DATA, BSS, ZDATA, ZBSS, ZDATA23, or ZBSS23 is specified when the -pid option has been specified
- By describing a section name in the symbol field of the .dseg directive, the data section can be named. If no section name is specified for a data section, the assembler automatically gives a default section name.
  The default section names of the data sections are shown below.
- The default section names have the relocation attributes shown above. Giving them any other attributes is not possible.

Example

```
test        .dseg   data
            .dw     0x1234
            .dw     0x5678
```

- The following characters are usable in section names.
  - Alphanumeric characters (0-9, a-z, A-Z)
  - Special characters (@, _, .)

.section

Indicate to the assembler the start of section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	`.section`	*section-name*`,` *relocation-attribute* `[ ,` `align=`*absolute-expressions*`]`	`[;` *comment*`]`

[Function]

- The .section directive indicates to the assembler the start of a section (no separation of code and data).

[Use]

- You can define all sections that can be defined via .cseg or .dseg directives using the .section directive, rather than differentiating code and data sections using the .cseg and .dseg directives.
- 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. [V2.03.00 or later]

.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.
- After the .org directive, it is valid until the next section definition directive.
- 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 each section starting at an absolute address takes the form of "section for which .org was written" + ".AT" + "specified address". The relocation attribute is the same as that of the section for which .org was 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".

[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
mov         r10, r11
.org        0x30                    ;"My_text.AT30" is allocated to address 0x30
mov         r11, r12
```

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         r10, r11                ;Allocated in "My_text.AT50"
```

[Caution]

- The operand value is in accordance with "Absolute expression". An illegal value will lead to an error and cause processing to end.
- The overall definition of a single section may contain multiple .org directives. Note, however, that an error will occur if an address specified for a section by .org is in an address range to which another section starting at an absolute address has been allocated in the same file.
- An error will occur in any of the following cases.
  - When the .org directive was written to the TDATA, PCTEXT, PCCONST16, PCCONST23 or PCCONST32 relocation attbibute section.
  - If the .org directive was written to the GP-relative section or EP-relative section when the -pid option has been specified

.offset

Specifies an offset from the first address of a section.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	`.offset`	*absolute-expression*	[`;` *comment*]

[Function]

- The .offset directive specifies an offset from the first address of a section that holds instruction code for the lines following the .offset directive.
- After the .org directive, it is valid until the next section definition directive.
- 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".
- Section names can also be enclosed in double-quotation marks (").

Example

```
.section    "My_data", data
.offset     0x12
mov         r10, r11            ; The offset is 0x12
```

[Caution]

- The operand value is in accordance with "Absolute expression". An illegal value will lead to an error and cause processing to end.
- 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 value specified as an operand of the .offset directive must be an Absolute expression in the range of 0x0 to 0x7fffffff.
  The actual value is limited by the memory size of the host machine where the program runs.
- The initial value for the area that is formed between the location of the .offset directive and the specified offset is 0x0.
- When the .offset directive is written in a section with relocation attribute "BSS", an error will occur.

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 5.10 Symbol Definition Directives

Directive	Overview
.set	Defines a name
.equ	Defines a symbol

.set

Defines a name.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
name	.set	*absolute-expression*	[; *comment*]

[Function]

Defines a symbol having a symbol name specified by the symbol field and 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]

- Incorrect formats for an operand will cause processing to end.
- The .set directive may be described anywhere in a source program.
- Each name is a redefinable symbol.
- Names cannot be externally defined.

[Example]

Defines the value of symbol sym1 as 0x10.

```
sym1    .set 0x10
```

[Caution]

- Any label reference or undefined symbol reference must not be used to specify a value.
  Otherwise, an error occurs.
- If a symbol of the same name as a label name or a macro name defined by the .macro directive is specified, an error occurs.

.equ

Defines a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
symbol	.equ	*absolute-expression*	[; *comment*]

[Function]

Defines a symbol having a symbol name specified by the symbol field and a absolute-expression value specified by the operand field.

[Use]

- You can use this directive to define symbols for numerical data that can be used instead of the actual numbers in the operands of machine-language instructions and directives in source code.

[Description]

- Incorrect formats for an operand will cause processing to end.
- The .set directive may be described anywhere in a source program.
- Symbols that have already been defined by using .equ cannot be redefined.
- The symbol generated by the .equ directive can be externally defined by the .public directive.

5.2.4 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 5.11 Compiler Output Directives

Directive	Overview
.file	Generates a symbol table entry
.line	Line-number information from the C source program
.stack	Defines the stack amount of consumption 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
.dbl_size [V2.04.00 or later]	Embeds the information of the -Xdbl_size option of the compiler into the object

.file

Generates a symbol table entry (FILE type).

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.file	"*file-name*"	[; *comment*]

[Function]

- The .file directive is compiler debugging information.

[Description]

- This is the name of the C source program file that the compiler outputs.

.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 .func 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.
- This is the line-number information of the C source program that the compiler outputs. A change made in the assembly source file is invalid.

.stack

Defines the stack size for a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	`.stack`	*symble-name*=*value*	[; *comment*]

[Function]

- Defines the stack size to be shown in Call Walker for a symbol.

[Description]

- This defines the stack size to be shown in Call Walker for a symbol.
- The stack size for a symbol can only be defined once, and subsequent definitions are ignored.
- The stack size can only be defined as a 4-byte range of 0x0 to 0xFFFFFFFC. If a different value is specified, then the definition is ignored.

[Remark]

- See the user's manual of CS+ for details about Call Walker.

._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 excluding $MACRO,$NOMACRO, $WARNING and $NOWARNING 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 directives (.db/.db2/.dhw/.db4/.dw/.db8/.ddw/.dshw/.ds/.float/.double)
  - macro directives (.macro/.irp/.rept/.local/.endm)
  - externally defined directive (.PUBLIC) [V1.05.00 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. Any other labels will lead to errors.
- Though instruction expansion is disabled in functions specified as inline_asm, instruction expansion can be enabled with the $MACRO control instruction. The effect of $MACRO and $NOWARNING is disabled when the ._line_end directive appears.

._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.

.dbl_size [V2.04.00 or later]

Embeds the information of the -Xdbl_size option of the compiler into the object.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	`.dbl_size`	*`size-value`*	`[; comment]`

[Function]

- The .dbl_size directive embeds the information of the -Xdbl_size option of the compiler into the object.

[Description]

- This directive embeds the information of the -Xdbl_size option of the compiler into the object.
- This directive does not affect the operation of the .float and .double directives.

5.2.5 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 5.12 Data Definition/Area Reservation Directives

Directive	Overview
.db	Initialization of byte area
.db2/.dhw	Initialization of 2-byte area
.dshw	Initializes a 2-byte area with the specified value, right-shifted one bit
.db4/.dw	Initialization of 4-byte area
.db8/.ddw	Initialization of 8-byte area
.float	Initialization of 4-byte area
.double	Initialization of 8-byte area
.ds	Allocates the memory area of the number of bytes specified by operand
.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*|"*string constant*"}[, ...]	[; *comment*]

[Function]

- The .db directive tells the assembler to initialize a memory area in byte units with the initial value(s) specified in the operand field.

[Use]

- Use the .db directive when defining an expression or character string used in the program.

[Description]

- The assembler initializes a byte area with:

  (a) Expression
  The value of an expression must be 1-byte data. Therefore, the value of the operand must be in the range of 0x0 to 0xFF. If the value exceeds 1 byte, the assembler will use only lower 1 byte of the value as valid data.

  (b) Character string constants
  If an operand is surrounded by double quotes ("), it is assumed to be a string constant.
  If a character string constants is described as the operand, an area of appropriate size will be reserved for each character in the string.

- Two or more initial values may be specified within a statement line of the .db directive.
- As an initial value, an expression that includes a relocatable symbol or external reference symbol may be described.
- 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
MASSAG:         .db     "ABCDEF"                ; (1)
DATA1:          .db     0xA, 0xB, 0xC           ; (2)
DATA3:          .db     "AB" + 1                ; (3)   <- Error
```

(1) A 6-byte area is initialized with character string 'ABCDEF'

(2) A 3-byte area is initialized with "0xA, 0xB, 0xC".

(3) This description occurs in an error.

.db2/.dhw

Initialization of 2-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.db2	*expression*[, ...]	[; *comment*]
[*label*:]	.dhw	*expression*[, ...]	[; *comment*]

[Function]

- The .db2 and .dhw directive tells the assembler to initialize a memory area in 2-byte units with the initial value(s) specified in the operand field.

[Use]

- Use the .db2 and .dhw directive when defining a 2-byte numeric constant such as an address or data used in the program.

[Description]

- The assembler initializes 2-byte area with:

  (a) Expression
  The value of an expression must be 2-byte data. Therefore, the value of the operand must be in the range of 0x0 to 0xFFFF. If the value exceeds 2-byte, the assembler will use only lower 2-byte of the value as valid data. No character string constants can be described as an initial value.

- If the relocation attribute of the section containing the .db2 and .dhw directive is "BSS", then an error is output because initial values cannot be specified.
- Two or more initial values may be specified within a statement line of the .db2 and .dhw directive.
- As an initial value, an expression that includes a relocatable symbol or external reference symbol may be described.

.dshw

Initializes a 2-byte area with the specified value, right-shifted one bit.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.dshw	*expression*[, ...]	[; *comment*]

[Function]

- Initializes a 2-byte area with the specified value, right-shifted one bit.

[Description]

- The value is allocated as 2-byte data, as the value of the expression right-shifted 1 bit.
- If the relocation attribute of the section is "BSS", then an error is output because the .dshw directive cannot be described.
- It is possible to code an absolute expression in the operand expression.
- The value of the expression, right-shifted one bit, must be in the range 0x0 to 0xFFF. In other cases, the data from the lower two bytes will be allocated.
- Any number of expressions may be specified on a single line, by separating them with commas.
- It is not possible to code string constants in the operand.

.db4/.dw

Initialization of 4-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.db4	*expression*[, ...]	[; *comment*]
[*label*:]	.dw	*expression*[, ...]	[; *comment*]

[Function]

- The .db4 and .dw directive tells the assembler to initialize a memory area in 4-byte units with the initial value(s) specified in the operand field.

[Use]

- Use the .db4 and .dw directive when defining a 4-byte numeric constant such as an address or data used in the program.

[Description]

- The assembler initializes 4-byte area with:

  (a) Expression
  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, the assembler will use only lower 2-byte of the value as valid data.
  No character string constants can be described as an initial value.

- Two or more initial values may be specified within a statement line of the .db4 and .dw directive.
- As an initial value, an expression that includes a relocatable symbol or external reference symbol may be described.
- If the relocation attribute of the section containing the .db4 and .dw directive is "BSS", then an error is output because initial values cannot be specified.

.db8/.ddw

Initialization of 8-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.db8	*absolute-expression*[, ...]	[; *comment*]
[*label*:]	.ddw	*absolute-expression*[, ...]	[; *comment*]

[Function]

- The .db8 and .ddw directive tells the assembler to initialize a memory area in 8-byte units with the initial value(s) specified in the operand field.

[Use]

- Use the .db8 and .ddw directive when defining a 8-byte numeric constant such as an address or data used in the program.

[Description]

- The assembler initializes 8-byte area with:

  (a) Expression
  The value of an expression must be 8-byte data. Therefore, the value of the operand must be in the range of 0x0 to 0xFFFFFFFFFFFFFFFF. If the value exceeds 8-byte, the assembler will use only lower 8-byte of the value as valid data.
  No character string constants can be described as an initial value.

- If the relocation attribute of the section is "BSS", then an error is output because the .db8 and .ddw directive cannot be described.
- Two or more initial values may be specified within a statement line of the .db8 and .ddw directive.

.float

Initialization of 4-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.float	*absolute-expression*[, ...]	[; *comment*]

[Function]

- The .float directive tells the assembler to initialize 4-byte area.
- The .float directive also tells the assembler to initialize a memory area in 4-byte units with the absolute-expression specified in the operand field.

[Description]

- The value of the absolute expression is allocated as a single-precision floating-point number. Consequently, the value of the expression must be between -3.40282347e+38 and 3.40282347e+38. In other cases, it is assumed as infinity with the same sign.
- If the relocation attribute of the section is "BSS", then an error is output because the .float directive cannot be described.
- Two or more absolute-expression may be specified within a statement line of the .float directive.

.double

Initialization of 8-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.double	*absolute-expression*[, ...]	[; *comment*]

[Function]

- The .double directive tells the assembler to initialize 8-byte area.
- The .double directive also tells the assembler to initialize a memory area in 8-byte units with the initial value(s) specified in the operand field.

[Description]

- The value of the absolute expression is allocated as a double-precision floating-point number. Consequently, the value of the expression must be between -1.7976931348623157e+308 and 1.7976931348623157e+308. In other cases, it is assumed as infinity with the same sign.
- If the relocation attribute of the section is "BSS", then an error is output because the .double directive cannot be described.
- Two or more absolute-expression may be specified within a statement line of the double directive.

.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]

- The .ds directive tells the assembler to reserve a memory area for the number of bytes specified in the operand field.

[Use]

- The .ds directive is mainly used to reserve a memory (RAM) area to be used in the program.
  If a label is specified, the value of the first address of the reserved memory area is assigned to the label. In the source module, this label is used for description to manipulate the memory.

[Description]

- If relocation attribute "BSS" is applied to the section where this instruction is written, an area for the number of bytes specified in the operand is allocated. For other sections, an area for the number of bytes specified in the operand is allocated and then initialized by 0.
  However, if the number of bytes for the size specification is 0, the area is not allocated.
- An absolute expression can be described as a size. If the size description is illegal, an error will occur.

.align

Aligns the value of the location counter.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.align	*line-condition*[, *absolute-expression*]	[; *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 first operand. If a hole results from aligning the value of the location counter, it is filled with the value of the absolute expression specified by the second operand, or with the default value of 0.
- Specify an even number of 2 or more, but less than 2^{31}, as the alignment condition. Otherwise, the CC-RH outputs the error message then stops assembling.
- The value of the second operand's absolute-expression must be in the range of 0x0 to 0xFF. If the value exceeds range of 0x0 to 0xFF, the assembler will use only lower 1-byte of the value as valid data.
- This directive merely aligns the value of the location counter in a specified file for the section. It does not align an address after arrangement.
- If this directive is written to a section with relocation attribute "BSS" and an absolute expression is specified, an error will occur.

5.2.6 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 5.13 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

.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	*label-name*[, *absolute-expression*]	[; *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]

- A label with the same name as the one specified by the first operand is declared as an external label[Note].
  Note that if a second operand was specified, this specifies the size of the data indicated by that label. However, specifications of size are ignored (although including them has been allowed to retain compatibility with CX).

  Note This is an external symbol (symbol with a GLOBAL binding class).

- Although this directive does not function any differently than an ".extern" directive in that it declares an external label, if this directive is used to declare a label with a definition in the specified file as an external label, use the ".extern" directive to declare labels without definitions in the specified file as external labels.
- The .public directive may be described anywhere in a source program.
- The ".public" directive can only define one symbol per line.
- When the symbol(s) to be described in the operand field isn't defined within the same module, an warning is output. When the symbol(s) isn't defined in any file, 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]

- Module 1

```
        .public A1              ; (a)
        .extern B1

A1:
        .db2    0x10

        .cseg   text
        jr      B1
```

- Module 2

```
        .public B1              ; (b)
        .extern A1
        .cseg   text
B1:
        mov     A1, r12
```

(a) This .public directive declares that symbol "A1" is to be referenced from other modules.

(b) This .public directive declares that symbol "B1" is 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.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.extern	*label-name*[, *absolute-expression*]	[; *comment*]

[Function]

- The .extern directive declares to the optimizing linker that a symbol in another module is to be referenced in this module.

[Use]

- To reference a symbol defined in another module, the .extern directive must be used to declare the symbol as an external definition.

[Description]

- A label with the same name as the one specified by the first operand is declared as an external label[Note].
  Note that if a second operand was specified, this specifies the size of the data indicated by that label. However, specifications of size are ignored (although including them has been allowed to retain compatibility with CX).

  Note This is an external symbol (symbol with a GLOBAL binding class).

- Although this directive does not function any differently than an ".public" directive in that it declares an external label, if this directive is used to declare a label without a definition in the specified file as an external label, use the ".public" directive to declare labels with definitions in the specified file as external labels.
- The .extern directive may be described anywhere in a source program.
- The ".extern" directive can only define one symbol per line.
- No error is output even if a symbol declared with the .extern directive is not referenced in the module.
- 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

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 5.14 Macro Directives

Directive	Overview
.macro	Executes a macro definition by assigning the macro name specified in the symbol field to a series of statements described between .macro directive and the .endm directive.
.local	The specified string is declared as a local symbol that will be replaced as a specific identifier.
.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 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

Executes a macro definition by assigning the macro name specified in the symbol field to a series of statements described between .macro directive and the .endm directive.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
macro-name	`.macro` : *Macro body* :	[*formal-parameter*[, ...]]	[; *comment*]

[Function]

- The .macro directive executes a macro definition by assigning the macro name specified in the symbol field to a series of statements (called a macro body) described between this directive and the .endm directive.

[Use]

- Define a series of frequently used statements in the source program with a macro name. After its definition only describe the defined macro name, and the macro body corresponding to the macro name is expanded.

[Description]

- If the .endm directive corresponding to .macro directive does not exist, an error occurs.
- For the macro name to be described in the symbol field, see the conventions of symbol description in "(2) Symbol".
- To call macro, describe the defined macro name in the mnemonic field. If a macro that is undefined at the point is called, an error occurs.
- For the formal parameter(s) to be described in the operand field, the same rules as the conventions of symbol description will apply.
- Formal parameters are valid only within the macro body.
- The number of formal parameters must be the same as the number of actual parameters. Otherwise, an error occurs.
- The maximum number of formal parameters that can be used depends on the amount of memory.
- The number of macros that can be defined within a single source module is not specifically limited. In other words, macros may be defined as long as there is memory space available.
- If a macro is called before it has been defined, an error will be output.
- The only actual parameters that can be specified in the macro call are label names, symbol names, numbers, registers, and instruction mnemonics.
  If a label expression (LABEL-1), a label beginning with a reference symbol (#LABEL), or base register specification ([gp]) or the like is specified, then a message will be output depending on the actual parameter specified, and assembly will halt.
- A line of a sentence can be designated in the macro-body. The part of a sentence, such as operand, cannot be allowed.
- An error will be output if a macro is defined in the macro body of a macro definition, but processing will continue (the content up to the corresponding ".endm" directive is ignored). Referencing a macro name will cause a definition error.

[Example]

```
ADMAC   .macro  PARA1, PARA2    ; (1)
        mov     PARA1, r12
        add     PARA2, r12
        .endm                   ; (2)

        ADMAC   0x10, 0x20      ; (3)
```

(1) A macro is defined by specifying macro name "ADMAC" and two formal parameters "PARA1" and "PARA2".

(2) This directive indicates the end of the macro definition.

(3) Macro "ADMAC" is referenced.

.local

The specified string is declared as a local symbol that will be replaced as a specific identifier.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	`.local`	*symbol-name*`[, ... ]`	`[;` *comment*`]`

[Function]

- The .local directive declares a specified symbol name as a local symbol that will be replaced as an assembler-specific symbol.

[Use]

- 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.

[Description]

- The maximum number of symbol names that can be used depends on the amount of memory.
- For a symbol defined by a label or symbol definition directive, the definition name is replaced with a name specific to each macro call.
- Only a label written after this directive within the macro body or a symbol defined by a symbol definition directive can be specified as the symbol name.
- The .local directive can be written in only a macro body, PEPT-ENDM block, IRP-ENDM block, or inline_asm function defined in a C source program. In any other case, an error will be output.
- If multiple local symbols are declared with the same name within a single block, an error will be output. Local symbols can be declared with the same name as long as they are in different blocks or they are within and without nested blocks.

[Example]

```
m1      .macro  x
        .local  a, b
        a:      .dw     a
        b:      .dw     x
.endm
m1      10
m1      20
```

The expansion is as follows.

```
.??00000000:    .dw     .??00000000
.??00000001:    .dw     10
.??00000002:    .dw     .??00000002
.??00000003:    .dw     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 expression specified in the operand field.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.rept :	*absolute-expression*	[; *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 expression specified in the operand field.

[Use]

- Use the .rept and .endm directives to describe a series of statements repeatedly in a source program.

[Description]

- An error occurs if the .rept directive is not paired with the .endm directive.
- If the .exitm directive appears in the REPT-ENDM block, subsequent expansion of the REPT-ENDM block is terminated at that location.
- Assembly control instructions may be described in the REPT-ENDM block.
- If a macro definition is written in the REPT-ENDM block, an error will be output.
- The value is evaluated as a 32-bit signed integer.
- If the result of evaluating the expression is negative, the assembler outputs the message then stops assembling.

[Example]

```
.cseg   text
        ; REPT-ENDM block
.rept   3               ; (1)
        nop
        ; Source text
.endm                   ; (2)
```

(1) This .rept directive tells the assembler to expand the REPT-ENDM block three consecutive times.

(2) This directive indicates the end of the REPT-ENDM block.

.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.

[Use]

- Use the .irp and .endm directives to describe a series of statements, only some of which become variables, repeatedly in a source program.

[Description]

- If the .endm directive corresponding to .irp directive does not exist, the assembler outputs the message.
- If the .exitm directive appears in the IRP-ENDM block, subsequent expansion of the IRP-ENDM block by the assembler is terminated.
- Macro definitions cannot be described in the IRP-ENDM block.
- Assembly control instructions may be described in the IRP-ENDM block.
- The maximum number of actual parameters that can be used depends on the amount of memory.

[Example]

```
.cseg   text

.irp    PARA 0xA, 0xB, 0xC              ; (1)
    ; IRP-ENDM block
        add     PARA, r12
        mov     r11, r12
.endm                                   ; (2)
    ; Source text
```

(1) The formal parameter is "PARA" and the actual parameters are the following three: "0xA", "0xB", and "0xC". This .irp directive tells the assembler to expand the IRP-ENDM block three times (i.e., the number of actual parameters) while replacing the formal parameter "PARA" with the actual parameters "0xA", "0xB", and "0xC"

(2) This directive indicates the end of the IRP-ENDM block.

.exitm

This directive skips the repetitive assembly of the .irp and .rept directives enclosing this directive at the innermost position.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.exitm		[; *comment*]

[Function]

- This directive skips the repetitive assembly of the .irp and .rept directives enclosing this directive at the innermost position.

[Description]

- If this directive is not enclosed by .irp and .rept directives, the assembler outputs the message then stops assembling.

.exitma

This directive skips the repetitive assembly of the irp and .rept directives enclosing this directive at the outermost position.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.exitma		[; *comment*]

[Function]

- This directive skips the repetitive assembly of the irp and .rept directives enclosing this directive at the outermost position.

[Description]

- If this directive is not enclosed by .irp and .rept directives, the assembler outputs the message then stops assembling.

.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.

[Use]

- The .endm directive must always be described at the end of a series of statements following the .macro, .rept, and/or the .irp directives.

[Description]

- A series of statements described between the .macro directive and .endm directive becomes a macro body.
- A series of statements described between the .rept directive and .endm directive becomes a REPT-ENDM block.
- A series of statements described between the .irp directive and .endm directive becomes an IRP-ENDM block.
- If the .macro, .rept, or .irp directive corresponding to this directive does not exist, the assembler outputs the message then stops assembling.

[Example]

(1) MACRO-ENDM

```
ADMAC   .macro  PARA1, PARA2
                mov     PARA1, r12
                add     PARA2, r12
        .endm
```

(2) REPT-ENDM

```
.rept   3
        add  1, r15
        sub  r15, r16
.endm
```

(3) IRP-ENDM

```
.irp    PARA 1, 2, 3
        add     PARA, r10
        st.w    r10, [r20]
.endm
```

5.3 Control Instructions

Control Instructions provide detailed instructions for assembler operation.

5.3.1 Outline

Control instructions provide detailed instructions for assembler operation and so are written in the source.
Control instructions do not become the target of object code generation.
Control instruction categories are displayed below.

Table 5.15 Control Instruction List

Control Instruction Type	Control Instruction
Assembler control instructions	REG_MODE, NOMACRO, MACRO, DATA, SDATA, NOWARNING, WARNING
File input control instructions	INCLUDE, BINCLUDE
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 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 5.16 Assembler Control Instructions

Control Instruction	Overview
REG_MODE	Outputs a register mode information
NOMACRO	Does not expand the subsequent instructions
MACRO	Cancels the specification made with the NOMACRO directive
DATA	Assumes that external data having symbol name extern_symbol has been allocated neither sdata nor sbss attribute section, and expands the instructions which reference that data
SDATA	Assumes that external data having symbol name extern_symbol has been allocated to the sdata or sbss attribute section, and dose not expand the instructions which reference that data
NOWARNING	Does not output warning messages
WARNING	Output warning messages

REG_MODE

A register mode information is output.

[Syntax]

```
[Δ]$[Δ]REG_MODE[Δ]specify-register-mode[Δ][;comment]
```

[Function]

- A register mode information is output into the object module file generated by the assembler.

[Description]

- Specify the register mode as "22" (indicating register mode 22); "32" (indicating register mode 32); or "common" (indicating universal register mode).
- A register mode information stores information about the number of working registers and register-variable registers used by the compiler. It is set in the object module file via this control instruction.
- If the register mode of this control instruction differs from the register mode specified via options, then the assembler will output a warning, and ignore the register mode specified via the options.
- If the register modes specified by this control instruction span multiple lines, and the specified register modes are different, then the first register-mode specification will be valid. The assembler will output warnings for the different register-mode specifications, and ignore those specifications.

NOMACRO

Does not expand the subsequent instructions.

[Syntax]

```
[Δ]$[Δ]NOMACRO[Δ][;comment]
```

[Function]

- Does not expand the subsequent instructions, other than the setfcond/jcond/jmp/cmovcond/sasfcond instructions.

MACRO

Cancels the specification made with the NOMACRO directive.

[Syntax]

```
[Δ]$[Δ]MACRO[Δ][;comment]
```

[Function]

- Cancels the specification made with the NOMACRO directive for the subsequent instructions.

DATA

An instruction that references external data of a symbol name is expanded into two instructions using gp.

[Syntax]

```
[Δ]$[Δ]DATA[Δ]extern_symbol[Δ][;comment]
```

[Function]

- An instruction that references external data of a symbol name is expanded into two instructions using gp.

SDATA

Assumes that external data having symbol name *extern_symbol* has been allocated to the sdata or sbss attribute section, and dose not expand the instructions which reference that data.

[Syntax]

```
[Δ]$[Δ]SDATA[Δ]extern_symbol[Δ][;comment]
```

[Function]

- Assumes that external data having symbol name *extern_symbol* has been allocated to the sdata or sbss attribute section, and does not expand the instructions which reference that data.

NOWARNING

Does not output warning messages.

[Syntax]

```
[Δ]$[Δ]NOWARNING[Δ][;comment]
```

[Function]

- Does not output warning messages for the subsequent instructions.

WARNING

Output warning messages.

[Syntax]

```
[Δ]$[Δ]WARNING[Δ][;comment]
```

[Function]

- Output warning messages for the subsequent instructions.

5.3.3 File input control instructions

Using the file input control instruction, the assembler can input an assembler source file or binary file to a specified position.

The following file input control instructions are available.

Table 5.17 File Input Control Instructions

Control Instruction	Overview
INCLUDE	Quotes a series of statements from another source module file
BINCLUDE	Inputs a binary file

INCLUDE

Quote a series of statements from another source module file.

[Syntax]

```
[Δ]$[Δ]INCLUDE[Δ]([Δ]file-name[Δ])[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.

[Use]

- A relatively large group of statements that may be shared by two or more source modules should be combined into a single file as an INCLUDE file.
  If the group of statements must be used in each source module, specify the filename of the required INCLUDE file with the INCLUDE control instruction.
  With this control instruction, you can greatly reduce time and labor in describing source modules.

[Description]

- The INCLUDE control instruction can only be described in ordinary source programs.
- 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) Standard include file folder
  (c) Folder in which the source file exists
  (d) Folder containing the (original) C source file
  (e) 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 assembler 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 it is not so closed, then an error occurs.

BINCLUDE

Inputs a binary file.

[Syntax]

```
[Δ]$[Δ]BINCLUDE[Δ]([Δ]file-name[Δ])[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]

- 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) Standard include file folder
  (c) Folder in which the source file exists
  (d) Folder containing the (original) C source file
  (e) Currently folder
- This control instruction handles the entire contents of the binary files. When a relocatable file is specified, this control instruction handles files configured in ELF format. Note that it is not just the contents of the .text selection, etc. that are handled.
- If a non-existent file is specified, the assembler outputs the message then stops assembling.

5.3.4 Conditional assembly control instructions

Using conditional assembly control instruction, the CC-RH can control the range of assembly according to the result of evaluating a conditional expression.

The following conditional assembly control instructions are available.

Table 5.18 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]IFDEF[Δ]switch-name[Δ][;comment]
```

[Function]

- If the switch name specified by the operand 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 specified switch name is not defined.
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

[Description]

- The rules of describing switch names are the same as the conventions of symbol description (for details, see "(2) Symbol").
- Switch names can overlap with user-defined symbols other than reserved words. Note, however, that overlapping between switch names is checked.
- Switch names are not output to the assembly list file's symbol-list information.

IFNDEF

Control based on symbol (assembly performed when the symbol is not defined).

[Syntax]

```
[Δ]$[Δ]IFNDEF[Δ]switch-name[Δ][;comment]
```

[Function]

- If the switch name specified by the operand is defined.
  Skips to the ELSEIF, ELSEIFN, ELSE, or ENDIF control instruction corresponding to this control instruction.

- If the specified 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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

[Description]

- The rules of describing switch names are the same as the conventions of symbol description (for details, see "(2) Symbol").
- Switch names can overlap with user-defined symbols other than reserved words. Note, however, that overlapping between switch names is checked.
- Switch names are not output to the assembly list file's symbol-list information.

IF

Control based on absolute expression (assembly performed when the value is true).

[Syntax]

```
[Δ]$[Δ]IF[Δ]absolute-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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

IFN

Control based on absolute expression (assembly performed when the value is false).

[Syntax]

```
[Δ]$[Δ]IFN[Δ]absolute-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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

ELSEIF

Control based on absolute expression (assembly performed when the value is true).

[Syntax]

```
[Δ]$[Δ]ELSEIF[Δ]absolute-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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

ELSEIFN

Control based on absolute expression (assembly performed when the value is false).

[Syntax]

```
[Δ]$[Δ]ELSEIFN[Δ]absolute-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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

ELSE

Control based on absolute expression/symbol.

[Syntax]

[Δ]$[Δ]ELSE[Δ]*absolute-expression*[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.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

ENDIF

End of control range.

[Syntax]

```
[Δ]$[Δ]ENDIF[Δ]absolute-expression[Δ][;comment]
```

[Function]

Indicates the end of the control range of a conditional assembly control instruction.

[Use]

- With these conditional assembly control instructions, source statements subject to assembly can be changed without major modifications to the source program.
- If a statement for debugging that becomes necessary only during the program development is described in a source program, whether or not the debugging statement should be assembled (translated into machine language) can be specified by setting switches for conditional assembly.

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".

```
PUSHMAC .macro  REG             ;The following two statements constitute the macro body.
        add     -4, sp
        st.w    REG, 0x0[sp]
.endm
```

If the following description is made after the above definition has been made, the macro is replaced by a code that "stores r19 in the stack".

```
PUSHMAC r19
```

In other words, the macro is expanded into the following codes.

```
add     -4, sp
st.w    r19, 0x0[sp]
```

5.4.3 Macro operator

This section describes the combination symbols "~" and "$", which are used to link strings in macros.

(1) ~ (Concatenation)

- The concatenation symbol concatenates "strings composed of numbers and characters similar to alphabet" with each other within a macro body.
  At macro expansion, the "strings" on the left and right sides of the concatenation symbol are concatenated and the concatenation symbol itself disappears after concatenating the strings.
- The concatenation symbol replaces formal parameters with arguments before concatenation.
- The character "~" can only be used as a combination symbol in a macro definition.
- The "~" in a character string and comment is simply handled as data.

Example 1.

```
abc     .macro  x
        abc~x:  mov     r10, r20
                sub def~x, r20
.endm
abc STU
```

```
[Development result]
abcSTU: mov     r10, r20
        sub     defSTU, r20
```

Example 2.

```
abc     .macro  x, xy
        a_~xy:  mov     r10, r20
        a_~x~y: mov     r20, r10
.endm
abc necel, STU
```

```
[Development result]
a_STU:  mov     r10, r20
a_stuy: mov     r20, r10
```

Example 3.

```
abc     .macro  x, xy
        ~ab:    mov r10, r20
.endm
abc stu, STU
```

```
[Development result]
ab:     mov     r10, r20
```

5.5 Reserved Words

The assembler has reserved words. Reserve word cannot be used in symbol, label, section name, macro name. If a reserved word is specified, the CC-RH outputs the message and stops assembling. Reserve 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
- Default section names

- GHS reserved sections ("_GHS", ".ghs", and section names starting with ".__ghs")

5.6 Predefined Macro Names

The assembler defines the following macros.

Predefined Macro Names	Remark
__RENESAS__	This value is set to 1.
__ASRH__	This value is set to 1.
__ASRH	This value is set to 1.
__RH850__	This value is set to 1.
__RH850	This value is set to 1.
__PIC	This value is set to 1. It is defined when the -pic option is specified.
__PIROD	This value is set to 1. It is defined when the -pirod option is specified.
__PID	This value is set to 1. It is defined when the -pid option is specified.

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 5.19 Assembler Generated Symbols

Symbol Name	Explanation
.??00000000 to .??FFFFFFFF	.local directive generation local symbols
.LMn_n (n : 0 - 4294967294)	Example : .LM0_1

5.8 Instruction Set

This section explains the instruction set supported by the CC-RH.

(1) Description of symbols
Next table lists the meanings of the symbols used further.

Table 5.20 Meaning of Symbols

Symbols	Meaning
CMD	Instruction
CMDi	Instruction(addi, mulhi, satsubi, andi, ori, xori)
reg, reg1, reg2	Register
r0, R0	Zero register
R1	Assembler-reserved register
gp	Global pointer (r4)
ep	Element pointer (r30)
[reg]	Base register
disp	Displacement (Displacement from the address) 32 bits unless otherwise stated.
disp*n*	*n*-bit displacement
imm	Immediate 32 bits unless otherwise stated.
imm*n*	*n*-bit immediate
bit#3	3-bit data for bit number specification
cc#3	3-bit data for specifying CC0 to CC7 (bits 24 to 31) of the FPSR floating-point system register
#label	Absolute address reference of label
label	Offset reference of label in section or PC offset reference
$label	gp offset reference of label
!label	Absolute address reference of label (without instruction expansion)
%label	ep offset reference of label (without instruction expansion)
HIGHW(*value*)	Higher 16 bits of *value*
LOWW(*value*)	Lower 16 bits of *value*
HIGHW1(*value*)	Higher 16 bits of *value* + bit value[Note] of bit number 15 of *value*
HIGH(*value*)	Upper 8 bits of the lower 16 bits of *value*
LOW(*value*)	Lower 8 bits of *value*
addr	Address
PC	Program counter
PSW	Program status word
regID	System register number (0 to 31)
selID	Group number (0 to 31)

Note The bit number 0 is LSB (Least Significant Bit).

(2) Operand

This section describes the description format of operand in assembler. In assembler, register, constant, symbol, label reference, and expression that composes of constant, symbol, label reference, operator and parentheses can be specified as the operands for instruction, and directives.

(a) Register

The registers that can be specified with the assembler are listed below.[Note]

r0, zero, r1, r2, hp, r3, sp, r4, gp, r5, tp, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, ep, r31, lp

Note For the ldsr and stsr instructions, the PSW, and system registers are specified by using the numbers. Further, in assembler, PC cannot be specified as an operand.

r0 and zero (Zero register), r2 and hp (Handler stack pointer), r3 and sp (Stack pointer), r4 and gp (Global pointer), r5 and tp (Text pointer), r30 and ep (Element pointer), r31 and lp (Link pointer) shows the same register.

(b) r0

r0 is the register which normally contains 0 value. This register does not substitute the result of an operation even if used as a destination register. Note that if machine instructions prohibit r0 from being specified as an operand, the assembler outputs the following message and stops assembling.

```
mov    0x10, r0
    ↓
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

(c) r1

The assembler-reserved register (r1) is used as a temporary register when instruction expansion is performed using the assembler. If r1 is specified as a source or destination register, the assembler outputs the following message[Note], then continues assembling.

Note Output of this message can be suppressed by specifying the warning message suppression option (-Xno_warning) upon starting the assembler.

```
mov    0x10, r1
    ↓
W0550013: r1 used as destination register.
```

The following instructions use r1 for instruction expansion:

ld.b, ld.h, ld.w, ld.bu, ld.hu, st.b, st.h, st.w, add, sddi, sub, subr, mulh, mulhi, mul, mulu, divh, div, divhu, divu, cmp, movea, cmov, satadd, satsub, satsubi, satsubr, or, ori, xor, xori, and, andi, not, tst, set1, clr1, not1, tst1, prepare, dispose

(d) Constants

As the constituents of the absolute expressions or relative expressions that can be used to specify the operands of the instructions and pseudo-instruction in the assembler, integer constants and character constants can be used.

Floating-point constants can be used to specify the operand of the .float and .double pseudo-instruction.

(e) Symbols

The assembler supports the use of symbols as the constituents of the absolute expressions or relative expressions that can be used to specify the operands of instructions and directives.

(f) Label reference

In assembler, label reference can be used as a component of available relative value as shown in operand designation of instruction/directive.

- Memory reference instruction (Load/store instruction, and bit manipulation instruction)
- Operation instruction (arithmetic operation instruction, saturated operation instruction, logical operation instruction)
- Branch instruction
- Area reservation directive

In assembler, the meaning of a label reference varies with the reference method and the differences used in the instructions/directives. Details are shown below.

Table 5.21 Label Reference

Reference Method	Instructions Used	Meaning
#label	Memory reference instruction, operation instruction and jmp instruction	The absolute address of the position at which the definition of label (*label*) exists (Offset from address 0[Note 1]). This has a 32-bit address and must be expanded into two instructions except ld23, st23, mov and jmp instruction.
	Area reservation directive	The absolute address of the position at which the definition of label (*label*) exists (Offset from address 0[Note 1]). Note that the 32-bit address is a value masked in accordance with the size of the area allocated.
!label	Memory reference instruction, operation instruction	The absolute address of the position at which the definition of label (*label*) exists (Offset from address 0 [Note 1]). This has a 16-bit address and cannot expand instructions if instructions with 16-bit displacement or immediate are specified. If any other instructions are specified, expansion into appropriate one instruction is possible. If the address defined by label (*label*) is not within a range expressible by 16 bits, an error will occur at the link time.
	Area reservation directive	The absolute address of the position at which the definition of label (*label*) exists (Offset from address 0 [Note 1]). Note that the 32-bit address is a value masked in accordance with the size of the area allocated.
label	Memory reference instruction, operation instruction	The offset in the section of the position where definition of the label (*label*) exists (offset from the initial address of the section where the definition of label (*label*) exists[Note 2]). This has a 32-bit offset and must be expanded into two instructions except ld23, st23 or mov instruction.
	Branch instruction except jmp instruction	The PC offset at the position where definition of label (*label*) exists (offset from the initial address of the instruction using the reference of label (*label*)).
	Area reservation directive	The offset in the section of the position where definition of the label (*label*) exists (offset from the initial address of the section where the definition of label (*label*) exists[Note 2]). Note that the 32-bit offset is a value masked in accordance with the size of the area allocated.
%label	Memory reference instruction, operation instruction	This has a 16-bit offset and cannot expand instructions if instructions with 16-bit displacement or immediate are specified. If any other instructions are specified, expansion into appropriate one instruction is possible. If the address defined by label (*label*) is not within a range expressible by 16 bits, an error will occur at the link time.
	Area reservation directive	The ep offset at the position where definition of the label (*label*) exists (offset from the address showing the element pointer). Note that the 32-bit offset is a value masked in accordance with the size of the area allocated.
$label	Memory reference instruction, operation instruction	The gp offset at the position where definition of the label (*label*) exists (offset from the address showing the global pointer).

Note 1. The offset from address 0 in object module file after link.

Note 2. The offset from the first address of the section (output section) in which the definition of label (*label*) exists is allocated in the linked object module file.

The meanings of label references for memory reference instructions, operation instructions, branch instructions, and area allocation pseudo-instruction are shown below.

Table 5.22 Memory Reference Instruction

Reference Method	Meaning
#label[reg]	The absolute address of label (*label*) is treated as a displacement. This has a 32-bit value and must be expanded into two instructions except ld23 or st23 instruction. By setting #label[r0], reference by an absolute address can be specified. Part of [reg] can be omitted. If omitted, the assembler assumes that [r0] has been specified.
label[reg]	The offset in the section of label (*label*) is treated as a displacement. This has a 32-bit value and must be expanded into two instructions except ld23 or st23 instruction. By specifying a register indicating the first address of section as reg and thereby setting label[reg], general register relative reference can be specified.
$label[reg]	The gp offset of label (*label*) is treated as a displacement. This has either a 32-bit or 16-bit value, from the section defined by label (*label*), and pattern of instruction expansion changes accordingly Note. If an instruction with a 16-bit value is expanded and the offset calculated from the address defined by label (*label*) is not within a range that can be expressed in 16 bits, an error is output at the link time. By setting $label [gp], relative reference of the gp register (called a gp offset reference) can be specified. Part of [reg] can be omitted. If omitted, the assembler assumes that [gp] has been specified.
!label[reg]	The absolute address of label (*label*) is treated as a displacement. This has a 16-bit value and instruction is not expanded. If the address defined by label (*label*) cannot be expressed in 16 bits, an error is output at the link time. By setting !lable[r0], reference by an absolute address can be specified. Part of [reg] can be omitted. If omitted, the assembler assumes that [r0] has been specified. However, unlike #label[reg] reference, instruction expansion is not executed.
%label[reg]	The offset from the ep symbol in the position where definition of the label (*label*) exists is treated as a displacement. This either has a 16-bit value, or depending on the instruction a value lower than this, and if it is not a value that can be expressed within this range, an error is output at the link time. Part of [reg] can be omitted. If omitted, the assembler assumes that [ep] has been specified.

Note See "(h) gp offset reference".

Table 5.23 Operation Instructions

Reference Method	Significance
#label	The absolute address of label (*label*) is treated as an immediate. This has a 32-bit value and must be expanded into two instructions.
label	The offset in the section of label (*label*) is treated as an immediate. This has a 32-bit value and must be expanded into two instructions.

Reference Method	Significance
$label	The gp offset of label (*label*) is treated as an immediate. This either has a 32-bit or 16-bit value, from the section defined by label (*label*), and pattern of instruction changes accordingly Note 1. If an instruction with a 16-bit value is expanded and the offset calculated from the address defined by label (*label*) is not within a range that can be expressed in 16 bits, an error is output at the link time.
!label	The absolute address of label (*label*) is treated as an immediate. This has a 16-bit value. If operation instruction for which a 16-bit value can be specify as an immediate are specified, and instruction is not expanded. If the value is not within a range that can be expressed in 16 bits, an error is output at the link time.
%label	The offset from the ep symbol in the position where definition of the label (*label*) exists is treated as an immediate. This has a 16-bit value. If operation instruction for which a 16-bit value can be specify as an immediate are specified, and instruction is not expanded. If the value is not within a range that can be expressed in 16 bits, an error is output at the link time.

Note 1. See "(h) gp offset reference".

Table 5.24 Branch Instructions

Reference Method	Meaning
#label	In jmp instruction, the absolute address of label (*label*) is treated as a jump destination address. This has a 32-bit value and must be expanded into two instructions.
label	In branch instructions other than the jmp instruction, PC offset of the label (*label*) is treated as a displacement. This has a 22-bit value, and if it is not within a range that can be expressed in 22 bits, an error is output at the link time.

Table 5.25 Area Reservation Directives

Reference Method	Meaning
#label !label	In .db4/.db2/.db directive, the absolute address of the label (*label*) is treated as a value. This has a 32-bit value, but is masked in accordance with the bit width of each directive.
label	In .db4/.db2/.db directive, the offset in the section defined by label (*label*) is treated as a value. This has a 32-bit value, but is masked in accordance with the bit width of each directive.
%label	The .db4, .db2, and .db directives treat the ep offset of label *label* as a value. This has a 32-bit value, but is masked in accordance with the bit width of each directive.
$label	The .db4, .db2, and .db directives treat the gp offset of label *label* as a value. This has a 32-bit value, but is masked in accordance with the bit width of each directive.

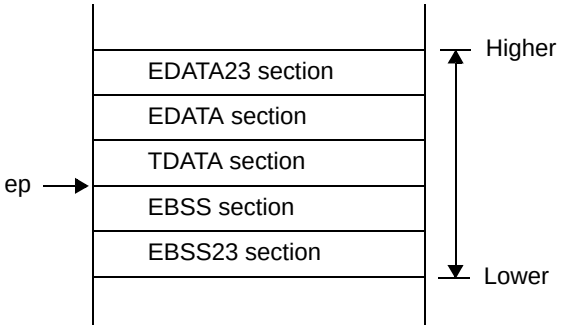
(g) ep offset reference

This section describes the ep offset reference. The CC-RH assumes that data explicitly stored in the sections with the following relocation attribute is shown below.

Reference through the offset from address indicated by the element pointer (ep).

- TDATA/TDATA4/TBSS4/TDATA5/TBSS5/TDATA7/TBSS7/TDATA8/TBSS8 section (Data is referred by memory reference instructions (sld/sst) of a small code size)
- EDATA/EBSS section (Data is referred by memory reference instructions (ld/st) of a large code size)
- EDATA23/EBSS23 section (Data is referred by memory reference instructions (ld23/st23) of a large code size)

Figure 5.2 Memory Location Image for ep Offset Reference Section



<1> Data allocation

In ep offset reference section, data is allocated to the sections as follows:

- When developing a program in C
  Allocate data by specifying the section attribute string starting with "ep_" in the "#pragma section" instruction.

- When developing a program in assembly language
  Data is allocated to the section of tdata, tdata4, tbss4, tdata5, tbss5, tdata7, tbss7, tdata8, tbss8, edata, ebss, edata23, or ebss23 relocation attribute sections by the section definition directives.

<2> Data reference
In cases where a reference via %label is made, the assembler generates a sequence of machine-language instructions to perform reference to the data at the corresponding ep offset.

Example

```
        .dseg   EDATA
sdata:  .db2    0xFFF0
        .dseg   DATA
data:   .db2    0xFFF0
        .cseg   TEXT
        ld.h    %sdata, r20     ; (1)
        ld.h    %data, r20      ; (2)
```

The assembler generates machine-language instructions that treat references via %label as ep-offset references in the cases of both (1) and (2).
The assembler assumes that the section in which the data is located is correct. As a result, it will not detect errors in data placement.

(h) gp offset reference
This section describes the gp offset reference. The CC-RH assumes that data stored in the sections with the following relocation attribute is basically shown below.

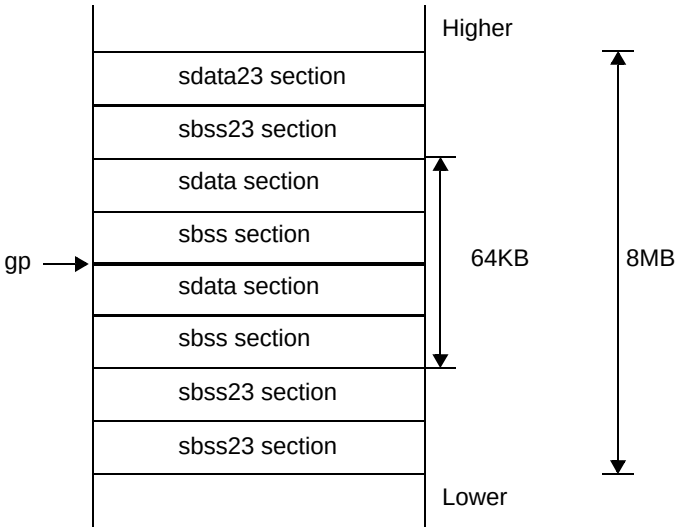
Referred by the offset from the address indicated by global pointer (gp).

- SDATA/SBSS section (Data is referred by memory reference instructions (ld/st) of a large code size)
- SDATA23/SBSS23 section (Data is referred by memory reference instructions (ld23/st23) of a large code size)

If r0-relative memory allocation for internal ROM or RAM is not done with the "#pragma section" command of C, or an assembly language section definition directive, all data is subject to gp offset reference.

<1> Data allocation

Figure 5.3 Memory Location Image for gp Offset Reference Section



Remark The sum of sdata- and sbss-attribute sections is 64 KB. gp is the center of the sdata section and sbss section.

In gp offset reference section, data is allocated to the sections as follows:
Explicitly allocate data that will be frequently referred to the sdata/sbss/sdata23/sbss23 sections. For allocation, use a section definition directive when using the assembly language, or the #pragma section command when using C.

<2> Data reference

- If the data has a definition in a specified file.

- If the data is to be allocated to the sdata or sbss section[Note].
  Generates a machine instruction that performs reference by using a16-bit displacement.
- If the data is not allocated to the sdata or sbss section.
  Generates a machine instruction string that performs reference by using a 32-bit displacement.

- If the data does not have a definition in a specified file.
  Assumes that the data is to be allocated to the sdata or sbss section (the label referenced by gp offset has a definition in the sdata/sbss section) and generates a machine instruction that performs reference by using a 16-bit displacement.

(i) About HIGH/LOW/HIGHW/LOWW/HIGHW1

<1> To refer memory by using 32-bit displacement
The assembler performs instruction expansion to refer the memory by using a 32-bit displacement, and generates an instruction string that performs the reference, by using the movhi and memory reference instructions and thereby constituting a 32-bit displacement from the higher 16 bits and lower 16 bits of the 32-bit displacement.

Example

`ld.w 0x18000[r11], r12`	`movhi HIGHW1(0x18000), r11, r1` `ld.w LOWW(0x18000)[r1], r12`

At this time, the memory reference instruction of machine instructions that uses the lower 16 bits as a displacement sign-extends the specified 16-bit displacement to a 32-bit value. To adjust the sign-extended bits, the assembler does not merely configure the displacement of the higher 16 bits by using the movhi instruction, instead it configures the following displacement.

Higher 16 bits + the most significant bit (bit of bit number 15) of the lower 16 bits

<2> HIGHW/LOWW/HIGHW1/HIGH/LOW
In the next table, the assembler can specify the higher 16 bits of a 32-bit value, the lower 16 bits of a 32-bit value, the value of the higher 16 bits + bit 15 of a 32-bit value, the higher 8 bits of a 16-bit value, and the lower 8 bits of a 16-bit value by using HIGHW, LOWW, HIGHW, HIGH, and LOW.[Note]

Note If this information cannot be internally resolved by the assembler, it is reflected in the relocation information and subsequently resolved by the link editor.

Table 5.26 Area Reservation Directives

HIGHW/LOWW/HIGHW1/ HIGH/LOW	Meaning
HIGHW (*value*)	Higher 16 bits of *value*
LOWW (*value*)	Lower 16 bits of *value*
HIGHW1 (*value*)	Higher 16 bits of *value* + bit value of bit number 15 of *value*
HIGH (*value*)	Upper 8 bits of the lower 16 bits of *value*
LOW (*value*)	Lower 8 bits of *value*

Example

```
        .dseg   DATA
L1:
            :
        .cseg   TEXT
        movhi   HIGHW ($L1), r0, r10  ; Stores the higher 16 bits of the gp
                                      ; offset value of L1 in the higher 16
                                     ; bits of r10, and the lower 16 bits to 0
        movea   LOWW ($L1), r0, r10   ; Sign-extends the lower 16 bits of the gp
                                      ; offset of L1 and stores to r10
            :
```

5.9 Extension of Assembly Language

For details on the assembler instructions supported by the assembler and machine instructions generated by the assembler, refer to the user's manual of each device.

This section describes specifications of instructions that have been changed from the device specifications or extended.

ld, st

- The device has the following ld and st instructions, and each instruction has a disp16 or disp23 operand.
  - (1) LD.B, LD.BU, LD.H, LD.HU, LD.W
  - (2) ST.B, ST.H, ST.W

  In asrh, the following mnemonics need to be specified when disp23 is to be explicitly specified for these instructions.
  - (1) ld23.b, ld23.bu, ld23.h, ld23.hu, ld23.w
  - (2) st23.b, st23.h, st23.w

- The device also has the following ld and st instructions.
  - (3) LD.DW
  - (4) ST.DW

  In asrh, these instructions can be specified in either format because the meaning is the same.
  - (3) ld.dw, ld23.dw
  - (4) st.dw, st23.dw

- If any of the following is specified for disp16, the assembler performs instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767, but within the range of -4,194,304 to +4,194,303

Format	Result of Assembly
`ld.w disp[reg1], reg2`	`ld.w disp23[reg1], reg2`

(b) Absolute expression having a value exceeding the range of -4,194,304 to +4,194,303

Format	Result of Assembly
`ld.w disp[reg1], reg2`	`movhi HIGHW1(disp), reg1, r1` `ld.w LOWW(disp)[r1], reg2`

(c) Relative expression having #label or label, or that having $label for a label having definition in the sdata/sbss-attribute section

Format	Result of Assembly
`ld.w #label[reg1], reg2`	`movhi HIGHW1(#label), reg1, r1` `ld.w LOWW(#label)[r1], reg2`
`ld.w label[reg1], reg2`	`movhi HIGHW1(label), reg1, r1` `ld.w LOWW(label)[r1], reg2`
`ld.w $label[reg1], reg2`	`movhi HIGHW1($label), reg1, r1` `ld.w LOWW($label)[r1], reg2`

- If disp is omitted, the assembler assumes 0.
- If an absolute expression, a relative expression having !label, a relative expression having #label, or a relative expression having #label and with LOWW applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression having $label, or a relative expression having $label and with LOWW applied, is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

- If a relative expression having %label is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [ep] is specified.

sld, sst

- The sld and sst instructions of the device should be specified in the following formats.
  - SLD.* dispN [ep], reg2
  - SST.* reg2, dispN [ep]

- In asrh, the format of %label should be used when specifying a relative expression in dispN.
- [ep] can be omitted.

```
sld.b  %_sym+2, r10          ; Same as "sld.b  %_sym+2[ep], r10"
```

add, mulh

- The add and mulh instructions of the device should be specified in the following formats.
  - ADD reg1, reg2
  - ADD imm5, reg2
  - MULH reg1, reg2
  - MULH imm5, reg2

- If the following is specified for imm in syntax "add imm, reg2" and "mulh imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`add imm16, reg`	`addi imm16, reg, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`add imm, reg`	`movhi HIGHW(imm), r0, r1` `add r1, reg`

Else

Format	Result of Assembly
`add imm, reg`	`mov imm, r1` `add r1, reg`

(c) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`add !label, reg`	`addi !label, reg, reg`
`add %label, reg`	`addi %label, reg, reg`
`add $label, reg`	`addi $label, reg, reg`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`add #label, reg`	`mov #label, r1` `add r1, reg`
`add label, reg`	`mov label, r1` `add r1, reg`
`add $label, reg`	`mov $label, r1` `add r1, reg`

addi, mulhi

- The addi and mulhi instructions of the device should be specified in the following formats.
  - ADDI imm16, reg1, reg2
  - MULHI imm16, reg1, reg2

- If the following is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

<1> If all the lower 16 bits of the value of imm are 0
If reg2 is r0 (Can be specified only in addi. An error will occur when specified in mulhi.)

Format	Result of Assembly
`addi imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `add reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`addi imm, reg1, reg2`	`movhi HIGHW(imm), r0, r1` `add r1, reg2`

Else

Format	Result of Assembly
`addi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `add reg1, reg2`

<2> Else
If reg2 is r0 (Can be specified only in addi. An error will occur when specified in mulhi.)

Format	Result of Assembly
`addi imm, reg1, r0`	`mov imm, r1` `add reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`addi imm, reg1, reg2`	`mov imm, r1` `add r1, reg2`

Else

Format	Result of Assembly
`addi imm, reg1, reg2`	`mov imm, reg2` `add reg1, reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section
If reg2 is r0 (Can be specified only in addi. An error will occur when specified in mulhi.)

Format	Result of Assembly
`addi #label, reg1, r0`	`mov #label, r1` `add reg1, r1`
`addi label, reg1, r0`	`mov label, r1` `add reg1, r1`
`addi $label, reg1, r0`	`mov $label, r1` `add reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`addi #label, reg1, reg2`	`mov #label, r1` `add r1, reg2`
`addi label, reg1, reg2`	`mov label, r1` `add r1, reg2`
`addi $label, reg1, reg2`	`mov $label, r1` `add r1, reg2`

Else

Format	Result of Assembly
`addi #label, reg1, reg2`	`mov #label, reg2` `add reg1, reg2`
`addi label, reg1, reg2`	`mov label, reg2` `add reg1, reg2`
`addi $label, reg1, reg2`	`mov $label, reg2` `add reg1, reg2`

adf, sbf, sasf, setf

- The adf, sbf, sasf, and setf instructions of the device should be specified in the following formats.
  - ADF cccc, reg1, reg2, reg3
  - SBF cccc, reg1, reg2, reg3
  - SASF cccc, reg2
  - SETF cccc, reg2

- In asrh, the following formats can also be used in addition to the above.
  - adf*cond* reg1, reg2, reg3
  - sbf*cond* reg1, reg2, reg3
  - sasf*cond* reg2
  - setf*cond* reg2

- Code that can be specified in *cond* and the result of assembly are shown in "Table 5.27 setfcond Instruction", using the setf instruction as an example.

Table 5.27 setf*cond* Instruction

Instruction	Flag Condition	Meaning of Flag Condition	Result of Assembly
`setfgt`	((S xor OV) or Z) = 0	Greater than (signed)	`setf 0xF`
`setfge`	(S xor OV) = 0	Greater than or equal (signed)	`setf 0xE`
`setflt`	(S xor OV) = 1	Less than (signed)	`setf 0x6`
`setfle`	((S xor OV) or Z) = 1	Less than or equal (signed)	`setf 0x7`
`setfh`	(CY or Z) = 0	Higher (Greater than)	`setf 0xB`
`setfnl`	CY = 0	Not lower (Greater than or equal)	`setf 0x9`
`setfl`	CY = 1	Lower (Less than)	`setf 0x1`
`setfnh`	(CY or Z) = 1	Not higher (Less than or equal)	`setf 0x3`
`setfe`	Z = 1	Equal	`setf 0x2`
`setfne`	Z = 0	Not equal	`setf 0xA`
`setfv`	OV = 1	Overflow	`setf 0x0`
`setfnv`	OV = 0	No overflow	`setf 0x8`
`setfn`	S = 1	Negative	`setf 0x4`
`setfp`	S = 0	Positive	`setf 0xC`
`setfc`	CY = 1	Carry	`setf 0x1`
`setfnc`	CY = 0	No carry	`setf 0x9`
`setfz`	Z = 1	Zero	`setf 0x2`
`setfnz`	Z = 0	Not zero	`setf 0xA`
`setft`	always 1	Always 1	`setf 0x5`
`setfsa`	SAT = 1	Saturated	`setf 0xD`Note

Note sa(0xD) cannot be specified in the adf or sbf instruction. If specified, an error will occur.

mul

- The mul instructions of the device should be specified in the following formats.
  - MUL reg1, reg2, reg3
  - MUL imm9, reg2, reg3

- If the instruction is executed in syntax "mul imm9, reg2, reg3", and the following expression is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression exceeding the range of -256 to +255, but within the range of -32,768 to +32,767

Format	Result of Assembly
`mul imm16, reg2, reg3`	`movea imm16, r0, r1` `mul r1, reg2, reg3`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`mul imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mul r1, reg2, reg3`

Else

Format	Result of Assembly
`mul imm, reg2, reg3`	`mov imm, r1` `mul r1, reg2, reg3`

(c) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`mul !label, reg2, reg3`	`movea !label, r0, r1` `mul r1, reg2, reg3`
`mul %label, reg2, reg3`	`movea %label, r0, r1` `mul r1, reg2, reg3`
`mul $label, reg2, reg3`	`movea $label, r0, r1` `mul r1, reg2, reg3`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`mul #label, reg2, reg3`	`mov #label, r1` `mul r1, reg2, reg3`
`mul label, reg2, reg3`	`mov label, r1` `mul r1, reg2, reg3`
`mul $label, reg2, reg3`	`mov $label, r1` `mul r1, reg2, reg3`

mulu

- The mulu instructions of the device should be specified in the following formats.
  - MULU reg1, reg2, reg3
  - MULU imm9, reg2, reg3

- If the instruction is executed in syntax "mulu imm9, reg2, reg3", and the following expression is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression within the range of -16 to -1

Format	Result of Assembly
`mulu imm5, reg2, reg3`	`mov imm5, r1` `mulu r1, reg2, reg3`

(b) Absolute expression exceeding the range of -16 to +511, but within the range of -32,768 to +32,767

Format	Result of Assembly
`mulu imm16, reg2, reg3`	`movea imm16, r0, r1` `mulu r1, reg2, reg3`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`mulu imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mulu r1, reg2, reg3`

Else

Format	Result of Assembly
`mulu imm, reg2, reg3`	`mov imm, r1` `mulu r1, reg2, reg3`

(d) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`mulu !label, reg2, reg3`	`movea !label, r0, r1` `mulu r1, reg2, reg3`
`mulu %label, reg2, reg3`	`movea %label, r0, r1` `mulu r1, reg2, reg3`
`mulu $label, reg2, reg3`	`movea $label, r0, r1` `mulu r1, reg2, reg3`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`mulu #label, reg2, reg3`	`mov #label, r1` `mulu r1, reg2, reg3`
`mulu label, reg2, reg3`	`mov label, r1` `mulu r1, reg2, reg3`
`mulu $label, reg2, reg3`	`mov $label, r1` `mulu r1, reg2, reg3`

divh

- The divh instructions of the device should be specified in the following formats.
  - DIVH reg1, reg2
  - DIVH reg1, reg2, reg3

- In asrh, the following formats can also be used in addition to the above.
  - divh imm, reg2
  - divh imm, reg2, reg3

- If the instruction is executed in syntax "divh imm, reg2", and the following expression is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value of other than 0 within the range of -16 to +15

Format	Result of Assembly
`divh imm5, reg`	`mov imm5, r1` `divh r1, reg`

(b) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`divh imm16, reg`	`movea imm16, r0, r1` `divh r1, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`divh imm, reg`	`movhi HIGHW(imm), r0, r1` `divh r1, reg`

Else

Format	Result of Assembly
`divh imm, reg`	`mov imm, r1` `divh r1, reg`

(d) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`divh !label, reg`	`movea !label, r0, r1` `divh r1, reg`
`divh %label, reg`	`movea %label, r0, r1` `divh r1, reg`
`divh $label, reg`	`movea $label, r0, r1` `divh r1, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`divh #label, reg`	`mov #label, r1` `divh r1, reg`
`divh label, reg`	`mov label, r1` `divh r1, reg`
`divh $label, reg`	`mov $label, r1` `divh r1, reg`

- If the instruction is executed in syntax "divh imm, reg2, reg3", and the following expression is specified for imm, the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

Format	Result of Assembly
`divh 0, reg2, reg3`	`divh r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

Format	Result of Assembly
`divh imm5, reg2, reg3`	`mov imm5, r1` `divh r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`divh imm16, reg2, reg3`	`movea imm16, r0, r1` `divh r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`divh imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `divh r1, reg2, reg3`

Else

Format	Result of Assembly
`divh imm, reg2, reg3`	`mov imm, r1` `divh r1, reg2, reg3`

(e) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`divh !label, reg2, reg3`	`movea !label, r0, r1` `divh r1, reg2, reg3`
`divh %label, reg2, reg3`	`movea %label, r0, r1` `divh r1, reg2, reg3`
`divh $label, reg2, reg3`	`movea $label, r0, r1` `divh r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`divh #label, reg2, reg3`	`mov #label, r1` `divh r1, reg2, reg3`
`divh label, reg2, reg3`	`mov label, r1` `divh r1, reg2, reg3`
`divh $label, reg2, reg3`	`mov $label, r1` `divh r1, reg2, reg3`

div, divhu, divu

- The div, divhu, and divu instructions of the device should be specified in the following formats.
  - DIV reg1, reg2, reg3
  - DIVHU reg1, reg2, reg3
  - DIVU reg1, reg2, reg3

- In asrh, the following formats can also be used in addition to the above.
  - div imm, reg2, reg3
  - divhu imm, reg2, reg3
  - divu imm, reg2, reg3

- If the following is specified for imm in syntax "div imm, reg2, reg3", "divhu imm,reg2,reg3", and "divu imm, reg2,reg3", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

Format	Result of Assembly
`div 0, reg2, reg3`	`div r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

Format	Result of Assembly
`div imm5, reg2, reg3`	`mov imm5, r1` `div r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`div imm16, reg2, reg3`	`movea imm16, r0, r1` `div r1, reg2, reg3`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`div imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `div r1, reg2, reg3`

Else

Format	Result of Assembly
`div imm, reg2, reg3`	`mov imm, r1` `div r1, reg2, reg3`

(e) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`div !label, reg2, reg3`	`movea !label, r0, r1` `div r1, reg2, reg3`
`div #label, reg2, reg3`	`movea #label, r0, r1` `div r1, reg2, reg3`
`div $label, reg2, reg3`	`movea $label, r0, r1` `div r1, reg2, reg3`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`div #label, reg2, reg3`	`mov #label, r1` `div r1, reg2, reg3`
`div label, reg2, reg3`	`mov label, r1` `div r1, reg2, reg3`
`div $label, reg2, reg3`	`mov $label, r1` `div r1, reg2, reg3`

cmp

- The cmp instructions of the device should be specified in the following formats.
  - CMP reg1, reg2
  - CMP imm5, reg2

- If the following is specified as imm in syntax "cmp imm, reg2", the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`cmp imm16, reg`	`movea imm16, r0, r1` `cmp r1, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`cmp imm, reg`	`movhi HIGHW(imm), r0, r1` `cmp r1, reg`

Else

Format	Result of Assembly
`cmp imm, reg`	`mov imm, r1` `cmp r1, reg`

(c) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`cmp !label, reg`	`movea !label, r0, r1` `cmp r1, reg`
`cmp %label, reg`	`movea %label, r0, r1` `cmp r1, reg`
`cmp $label, reg`	`movea $label, r0, r1` `cmp r1, reg`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`cmp #label, reg`	`mov #label, r1` `cmp r1, reg`
`cmp label, reg`	`mov label, r1` `cmp r1, reg`
`cmp $label, reg`	`mov $label, r1` `cmp r1, reg`

mov

- The mov instruction of the device should be specified in the following formats.
  - MOV reg1, reg2
  - MOV imm5, reg2
  - MOV imm32, reg1

- In asrh, the following format can also be used in addition to the above.
  - mov32 imm32, reg1

  This is specified when a 48-bit instruction with a 32-bit imm is to be explicitly used.

- If the instruction is executed in syntax "mov imm, reg2", and reg2 is r0, the assembler generates a 48-bit mov machine instruction.

- If the following is specified as imm in syntax "mov imm, reg2" and reg2 is not r0, the assembler executes instruction expansion to generate one machine instruction.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`mov imm16, reg`	`movea imm16, r0, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`mov imm, reg`	`movhi HIGHW(imm), r0, reg`

Else[Note]

Format	Result of Assembly
`mov imm, reg`	`mov imm32, reg`

Note A 16-bit mov instruction is replaced by a 48-bit mov instruction.

(c) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`mov !label, reg`	`movea !label, r0, reg`
`mov %label, reg`	`movea %label, r0, reg`
`mov $label, reg`	`movea $label, r0, reg`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section[Note]

Format	Result of Assembly
`mov #label, reg`	`mov #label, reg`
`mov label, reg`	`mov label, reg`
`mov $label, reg`	`mov $label, reg`

Note A 16-bit mov instruction is replaced by a 48-bit mov instruction.

movea

- The movea instruction of the device should be specified in the following formats.
  - MOVEA imm16, reg1, reg2

- If the following is specified for imm, the assembler executes instruction expansion to generate one or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`movea imm, reg1, reg2`	`movhi HIGHW(imm), reg1, reg2`

Else

Format	Result of Assembly
`movea imm, reg1, reg2`	`movhi HIGHW1(imm), reg1, r1` `movea LOWW(imm), r1, reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`movea #label, reg1, reg2`	`movhi HIGHW1(#label), reg1, r1` `movea LOWW(#label), r1, reg2`
`movea label, reg1, reg2`	`movhi HIGHW1(label), reg1, r1` `movea LOWW(label), r1, reg2`
`movea $label, reg1, reg2`	`movhi HIGHW1($label), reg1, r1` `movea LOWW($label), r1, reg2`

cmov

- The cmov instruction of the device should be specified in the following formats.
  - CMOV cccc, reg1, reg2, reg3
  - CMOV cccc, imm5, reg2, reg3

- In asrh, the following formats can also be used in addition to the above.
  - cmov*cond* reg1, reg2, reg3
  - cmov*cond* imm, reg2, reg3

  Code that can be specified in *cond* and the result of assembly are the same as those for the setf instruction. For details, see "Table 5.27 setfcond Instruction".

- If the following is specified as imm in syntax "cmov cccc, imm, reg2, reg3" or "cmov*cond* imm, reg2, reg3", the assembler executes instruction expansion to generate multiple machine instructions.

  (a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`cmov imm4, imm16, reg2, reg3`	`movea imm16, r0, r1` `cmov imm4, r1, reg2, reg3`

  (b) Absolute expression having a value exceeding the range of -32,768 to +32,767
  If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`cmov imm4, imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `cmov imm4, r1, reg2, reg3`

  Else

Format	Result of Assembly
`cmov imm4, imm, reg2, reg3`	`mov imm, r1` `cmov imm4, r1, reg2, reg3`

  (c) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`cmov imm4, #label, reg2, reg3`	`mov #label, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, label, reg2, reg3`	`mov label, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, $label, reg2, reg3`	`mov $label, r1` `cmov imm4, r1, reg2, reg3`

(d) Relative expression having !label or %label, or that having $label for a label with a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`cmov imm4, !label, reg2, reg3`	`movea !label, r0, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, %label, reg2, reg3`	`movea %label, r0, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, $label, reg2, reg3`	`movea $label, r0, r1` `cmov imm4, r1, reg2, reg3`

satadd

- The satadd instructions of the device should be specified in the following formats.
  - SATADD reg1, reg2
  - SATADD imm5, reg2
  - SATADD reg1, reg2, reg3

- If the following is specified for imm in syntax "satadd imm, reg2", the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`satadd imm16, reg`	`movea imm16, r0, r1` `satadd r1, reg`

(b) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`satadd imm, reg`	`movhi HIGHW(imm), r0, r1` `satadd r1, reg`

Else

Format	Result of Assembly
`satadd imm, reg`	`mov imm, r1` `satadd r1, reg`

(c) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`satadd !label, reg`	`movea !label, r0, r1` `satadd r1, reg`
`satadd %label, reg`	`movea %label, r0, r1` `satadd r1, reg`
`satadd $label, reg`	`movea $label, r0, r1` `satadd r1, reg`

(d) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`satadd #label, reg`	`mov #label, r1` `satadd r1, reg`
`satadd label, reg`	`mov label, r1` `satadd r1, reg`
`satadd $label, reg`	`mov $label, r1` `satadd r1, reg`

satsub

- The satsub instructions of the device should be specified in the following formats.
  - SATSUB reg1, reg2
  - SATSUB reg1, reg2, reg3

- In asrh, the following formats can also be used in addition to the above.
  - satsub imm, reg2

- If the instruction is executed in syntax "satsub imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

Format	Result of Assembly
`satsub 0, reg`	`satsub r0, reg`

(b) Absolute expression having a value in the range of -32,768 to +32,767

Format	Result of Assembly
`satsub imm16, reg`	`satsubi imm16, reg, reg`

(c) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`satsub imm, reg`	`movhi HIGHW(imm), r0, r1` `satsub r1, reg`

Else

Format	Result of Assembly
`satsub imm, reg`	`mov imm, r1` `satsub r1, reg`

(d) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`satsub !label, reg`	`satsubi !label, reg, reg`
`satsub %label, reg`	`satsubi %label, reg, reg`
`satsub $label, reg`	`satsubi $label, reg, reg`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`satsub #label, reg`	`mov #label, r1` `satsub r1, reg`
`satsub label, reg`	`mov label, r1` `satsub r1, reg`
`satsub $label, reg`	`mov $label, r1` `satsub r1, reg`

satsubi

- The satsubi instructions of the device should be specified in the following formats.
  - SATSUBI imm16, reg1, reg2

- If the following is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

<1> If all the lower 16 bits of the value of imm are 0
If reg2 is the same as reg1

Format	Result of Assembly
`satsubi imm, reg1, reg2`	`movhi HIGHW(imm), r0, r1` `satsub r1, r2`

Else

Format	Result of Assembly
`satsubi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `satsubr reg1, reg2`

<2> Else
If reg2 is the same as reg1

Format	Result of Assembly
`satsubi imm, reg1, reg2`	`mov imm, r1` `satsub r1, reg2`

Else

Format	Result of Assembly
`satsubi imm, reg1, reg2`	`mov imm, reg2` `satsubr reg1, reg2`

(b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section
If reg2 is the same as reg1

Format	Result of Assembly
`satsubi #label, reg1, reg2`	`mov #label, r1` `satsub r1, reg2`
`satsubi label, reg1, reg2`	`mov label, reg2` `satsub r1, reg2`
`satsubi $label, reg1, reg2`	`mov $label, reg2` `satsub r1, reg2`

Else

Format	Result of Assembly
`satsubi #label, reg1, reg2`	`mov #label, reg2` `satsubr reg1, reg2`
`satsubi label, reg1, reg2`	`mov label, reg2` `satsubr reg1, reg2`
`satsubi $label, reg1, reg2`	`mov $label, reg2` `satsubr reg1, reg2`

and, or, xor

- The and, or, and xor instructions of the device should be specified in the following formats.
  - AND reg1, reg2
  - OR reg1, reg2
  - XOR reg1, reg2

- In asrh, the following formats can also be used in addition to the above.
  - and imm, reg2
  - or imm, reg2
  - xor imm, reg2

- If the following is specified for imm in syntax "and imm, reg2", "or imm, reg2", and "xor imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

Format	Result of Assembly
`and 0, reg`	`and r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

Format	Result of Assembly
`and imm16, reg`	`andi imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

Format	Result of Assembly
`and imm5, reg`	`mov imm5, r1` `and r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

Format	Result of Assembly
`and imm16, reg`	`movea imm16, r0, r1` `and r1, reg`

(e) Absolute expression exceeding the above ranges
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`and imm, reg`	`movhi HIGHW(imm), r0, r1` `and r1, reg`

Else

Format	Result of Assembly
`and imm, reg`	`mov imm, r1` `and r1, reg`

(f) Relative expression having !label or %label

Format	Result of Assembly
`and !label, reg`	`andi !label, reg, reg`
`and %label, reg`	`andi %label, reg, reg`

(g) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`and $label, reg`	`movea $label, r0, r1` `and r1, reg`

(h) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`and #label, reg`	`mov #label, r1` `and r1, reg`
`and label, reg`	`mov label, r1` `and r1, reg`
`and $label, reg`	`mov $label, r1` `and r1, reg`

andi, ori, xori

- The andi, ori, and xori instructions of the device should be specified in the following formats.
  - ANDI imm16, reg1, reg2
  - ORI imm16, reg1, reg2
  - XORI imm16, reg1, reg2

- If the following is specified for imm, the assembler executes instruction expansion to generate multiple machine instructions.

(a) Absolute expression having a value in the range of -16 to -1
If reg2 is r0

Format	Result of Assembly
`andi imm5, reg1, r0`	`mov imm5, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi imm5, reg1, reg2`	`mov imm5, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi imm5, reg1, reg2`	`mov imm5, reg2` `and reg1, reg2`

(b) Absolute expression having a value in the range of -32,768 to -17
If reg2 is r0

Format	Result of Assembly
`andi imm16, reg1, r0`	`movea imm16, r0, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi imm16, reg1, reg2`	`movea imm16, r0, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi imm16, reg1, reg2`	`movea imm16, r0, reg2` `and reg1, reg2`

(c) Absolute expression exceeding the above ranges

<1> If all the lower 16 bits of the value of imm are 0
If reg2 is r0

Format	Result of Assembly
`andi imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi imm, reg1, reg2`	`movhi HIGHW(imm), r0, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `and reg1, reg2`

<2> Else
If reg2 is r0

Format	Result of Assembly
`andi imm, reg1, r0`	`mov imm, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi imm, reg1, reg2`	`mov imm, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi imm, reg1, reg2`	`mov imm, reg2` `and reg1, reg2`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section
If reg2 is r0

Format	Result of Assembly
`andi $label, reg1, r0`	`movea $label, r0, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi $label, reg1, reg2`	`movea $label, r0, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi $label, reg1, reg2`	`movea $label, r0, reg2` `and reg1, reg2`

(e) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section
If reg2 is r0

Format	Result of Assembly
`andi #label, reg1, r0`	`mov #label, r1` `and reg1, r1`
`andi label, reg1, r0`	`mov label, r1` `and reg1, r1`
`andi $label, reg1, r0`	`mov $label, r1` `and reg1, r1`

Else if reg2 is the same as reg1

Format	Result of Assembly
`andi #label, reg1, reg2`	`mov #label, r1` `and r1, reg2`
`andi label, reg1, reg2`	`mov label, r1` `and r1, reg2`
`andi $label, reg1, reg2`	`mov $label, r1` `and r1, reg2`

Else

Format	Result of Assembly
`andi #label, reg1, reg2`	`mov #label, reg2` `and reg1, reg2`
`andi label, reg1, reg2`	`mov label, reg2` `and reg1, reg2`
`andi $label, reg1, reg2`	`mov $label, reg2` `and reg1, reg2`

not, satsubr, sub, subr, tst

- The not, satsubr, sub, subr, and tst instructions of the device should be specified in the following formats.
  - NOT reg1, reg2
  - SATSUBR reg1, reg2
  - SUB reg1, reg2
  - SUBR reg1, reg2
  - TST reg1, rg2

- In asrh, the following formats can also be used in addition to the above.
  - not imm, reg2
  - satsubr imm, reg2
  - sub imm, reg2
  - subr imm, reg2
  - tst imm, reg2

- If the following is specified for imm in syntax "not imm, reg2", "satsubr imm, reg2", "sub imm, reg2", and "subr imm, reg2", "tst imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

Format	Result of Assembly
`not 0, reg`	`not r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

Format	Result of Assembly
`not imm5, reg`	`mov imm5, r1` `not r1, reg`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

Format	Result of Assembly
`not imm16, reg`	`movea imm16, r0, r1` `not r1, reg`

(d) Absolute expression having a value exceeding the range of -32,768 to +32,767
If all the lower 16 bits of the value of imm are 0

Format	Result of Assembly
`not imm, reg`	`movhi HIGHW(imm), r0, r1` `not r1, reg`

Else

Format	Result of Assembly
`not imm, reg`	`mov imm, r1` `not r1, reg`

(e) Relative expression having !label or %label, or that having $label for a label having a definition in the sdata/sbss-attribute section

Format	Result of Assembly
`not !label, reg`	`movea !label, r0, r1` `not r1, reg`
`not %label, reg`	`movea %label, r0, r1` `not r1, reg`
`not $label, reg`	`movea $label, r0, r1` `not r1, reg`

(f) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`not #label, reg`	`mov #label, r1` `not r1, reg`
`not label, reg`	`mov label, r1` `not r1, reg`
`not $label, reg`	`mov $label, r1` `not r1, reg`

b*cond*

- The b*cond* instructions of the device should be specified in the following formats.
  - B*cond* disp9
  - B*cond* disp17

- In asrh, the following formats can also be used in addition to the above.
  - b*cond*9 disp9
  - b*cond*17 disp17
  - j*cond* disp9
  - j*cond*9 disp9
  - j*cond*17 disp17
  - jbr disp

- b*cond*9 or b*cond*17 should be specified when output should be with a fixed disp width at all times. These instructions will not be expanded.
- j*cond*, j*cond*9, and j*cond*17 have the same meaning as b*cond*, b*cond*9, and b*cond*17, respectively.
- jbr has the same meaning as br.
- Code that can be specified in *cond* is shown in "Table 5.28 bcond Instruction".

Table 5.28 b*cond* Instruction

Instruction	Flag Condition	Meaning of Flag Condition
bgt	((S xor OV) or Z) = 0	Greater than (signed)
bge	(S xor OV) = 0	Greater than or equal (signed)
blt	(S xor OV) = 1	Less than (signed)
ble	((S xor OV) or Z) = 1	Less than or equal (signed)
bh	(CY or Z) = 0	Higher (Greater than)
bnl	CY = 0	Not lower (Greater than or equal)
bl	CY = 1	Lower (Less than)
bnh	(CY or Z) = 1	Not higher (Less than or equal)
be	Z = 1	Equal
bne	Z = 0	Not equal
bv	OV = 1	Overflow
bnv	OV = 0	No overflow
bn	S = 1	Negative
bp	S = 0	Positive
bc	CY = 1	Carry
bnc	CY = 0	No carry
bz	Z = 1	Zero
bnz	Z = 0	Not zero

Instruction	Flag Condition	Meaning of Flag Condition
br	---	Always (Unconditional)
bsa	SAT = 1	Saturated

- The bt and bf instructions cannot be specified in asrh.
- If the following is specified for disp22, the assembler executes instruction expansion and generates one or more machine-language instructions.

(a) Absolute expression having a value exceeding the range of -256 to +255 but within the range of -65,536 to +65,535 or a relative expression having a PC offset reference of label with a definition in the same section of the same file as this instruction and having a value exceeding the range of -256 to +255 but within the range of -65,536 to +65,535

Format	Result of Assembly
`br disp17`	`jr disp17`
`bcond disp17`	`bcond disp17`

(b) Absolute expression having a value exceeding the range of -65,536 to +65,535 but within the range of -2,097,150 to +2,097,153[Note 1], a relative expression having a PC offset reference of label with a definition in the same section of the same file as this instruction and having a value exceeding the range of -65536 to +65535, or a relative expression having a PC offset reference of label without a definition in the same file or section as this instruction.

Format	Result of Assembly
`br disp22`	`jr disp22`

`bsa disp22`	`bsa Label1`
	`br Label2`
	`Label1:`
	`jr disp22 - 4`
	`Label2:`

`bcond disp22`	`bncond Label`[Note 2]
	`jr disp22 - 2`
	`Label:`

Note 1. The range of -2,097,150 to +2,097,153 applies to instructions other than br and bsa. The range for the br instruction is from -2,097,152 to +2,097,151, and that for the bsa instruction is from -2,097,148 to +2,097,155.

Note 2. b*ncond* denotes an instruction that effects control of branches under opposite conditions, for example, bnz for bz or ble for bgt.

jmp

- The jmp instructions of the device should be specified in the following formats.
  - JMP [reg1]
  - JMP disp32 [reg1]

- In asrh, the following formats can also be used in addition to the above.

Format	Meaning
`jmp disp32`	`jmp disp32[r0]`
`jmp32 [reg1]`	`jmp [reg1]`
`jmp32 disp32`	`jmp disp32[r0]`
`jmp32 disp32[reg1]`	`jmp disp32[reg1]`

jr

- The jr instructions of the device should be specified in the following formats.
  - JR disp22
  - JR disp32

- In asrh, the following formats can also be used in addition to the above.
  - jr22 disp22
  - jr32 disp32

- jr instructions are interpreted as disp22 when the -Xasm_far_jump option is not specified, or disp32 when the -Xasm_far_jump option is specified.
- jr22 or jr32 should be specified when output should be with a fixed disp width at all times. These instructions will not be affected by the -Xasm_far_jump option.

jarl

- The jarl instructions of the device should be specified in the following formats.
  - JARL disp22, reg2
  - JARL disp32, reg1
  - JARL [reg1], reg3

- In asrh, the following formats can also be used in addition to the above.
  - jarl22 disp, reg2
  - jarl32 disp, reg1

- jarl instructions are interpreted as disp22 when the -Xasm_far_jump option is not specified, or disp32 when the -Xasm_far_jump option is specified.
- jarl22 or jarl32 should be specified when output should be with a fixed disp width at all times. These instructions will not be affected by the -Xasm_far_jump option.

set1, clr1, not1, tst1

- The set1, clr1, not1, and tst1 instructions of the device should be specified in the following formats.
  - set1 bit#3, disp16 [reg1]
  - clr1 bit#3, disp16 [reg1]
  - not1 bit#3, disp16 [reg1]
  - tst1 bit#3, disp16 [reg1]
  - set1 reg2, [reg1]
  - clr1 reg2, [reg1]
  - not1 reg2, [reg1]
  - tst1 reg2, [reg1]

- Among the above formats, the format of "*op* bit#3, disp16 [reg1]" will be interpreted by asrh as follows:
  - If any of the following is specified as disp16, the assembler executes instruction expansion to generate multiple machine instructions.
    - Absolute expression having a value exceeding the range of -32,768 to +32,767

Format	Result of Assembly
`set1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `set1 bit#3, LOWW(disp)[r1]`

    - Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

Format	Result of Assembly
`set1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `set1 bit#3, LOWW(disp)[r1]`

  - If disp is omitted, the assembler assumes 0.
  - If a relative expression with #label, or a relative expression with #label and with LOWW applied is specified as disp, [reg1] can be omitted. In that case, it is assumed that [r0] was specified.
  - If a relative expression with $label, or a relative expression with $label and with LOWW applied is specified as disp, [reg1] can be omitted. In that case, it is assumed that [gp] was specified.
  - If a relative expression with %label, or a relative expression with %label and with LOWW applied is specified as disp, [reg1] can be omitted. In that case, it is assumed that [ep] was specified.

push, pushm, pop, popm

- The push, pushm, pop, and popm instructions are directives that are not available in the device.
  They push the values of specified registers to the stack area or pop the values of specified registers from the stack area.

- In asrh, these instructions should be specified in the following formats.
  - push reg1
  - pushm reg1, reg2, ..., regN
  - pop reg1
  - popm reg1, reg2, ..., regN

- The instructions are expanded as shown below and multiple machine-language instructions are generated.

Format	Result of Assembly
`push reg1`	`add -4, sp` `st.w reg1, 0[sp]`

`pushm reg1, reg2, …, regN`	`addi -4 * N, sp, sp`
	`st.w regN, 4 * (N - 1)[sp]`
	`:`
	`st.w reg2, 4 * 1[sp]`
	`st.w reg1, 4 * 0[sp]`

`pop reg1`	`ld.w 0[sp], reg1` `add 4, sp`

`popm reg1, reg2, …, regN`	`ld.w 4 * 0[sp], reg1`
	`ld.w 4 * 1[sp], reg2`
	`:`
	`ld.w 4 * (N - 1)[sp], regN`
	`addi 4 * N, sp, sp`

Note The values of PSW are undefined because the add or addi instruction is used.
We recommend the use of the pushsp or popsp instruction provided by the device.

prepare, dispose

- The prepare and dispose instructions of the device should be specified in the following formats.
  - PREPARE list12, imm5
  - PREPARE list12, imm5, sp/imm
  - DISPOSE imm5, list12
  - DISPOSE imm5, list12, [reg1]

- In asrh, they need to be specified in the following formats.
  - prepare list, imm1
  - prepare list, imm1, imm2
  - prepare list, imm1, sp
  - dispose imm1, list
  - dispose imm1, list, [reg1]

- list specifies the 12 registers that can be manipulated by the prepare or dispose instruction. The following can be specified as list.
  - Register
    Specify the registers (r20 to r31) to be pushed, delimiting each with a comma.
  - Absolute expression having a value of up to 12 bits
    The 12 bits and 12 registers correspond as follows:

bit 11 bit 0

r30	r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31

[Example]

```
prepare r26, r29, r31, 0x10                ; Same as "prepare 0x103, 0x10"
```

- Specify the increased amount of sp by a multiple of four in imm1. The assembler shifts that value to the right by 2 bits and stores it in a machine instruction. The device will shift the value to the left by 2 bits prior to execution.

[Example]
To allocate 16 bytes of stack area, specify 0x10 instead of 0x4.

```
prepare 0, 0x10
```

- When the following is specified as imm1, the assembler executes instruction expansion to generate two or more machine instructions.
  If the instruction is executed in syntax "prepare list, imm1, sp", an absolute expression having a value exceeding the range of 0 to 127 can not be specified for imm1.

(a) Absolute expression having a value exceeding the range of 0 to 127, but within the range of 0 to 32,767

Format	Result of Assembly
`prepare list, imm1`	`prepare list, 0` `movea -imm1, sp, sp`
`prepare list, imm1, imm2`	`prepare list, 0, imm2` `movea -imm1, sp, sp`
`dispose imm1, list`	`movea imm1, sp, sp` `dispose 0, list`
`dispose imm1, list, [reg1]`	`movea imm1, sp, sp` `dispose 0, list, [reg1]`

(b) Absolute expression having a value exceeding the range of 0 to 32,767

Format	Result of Assembly

`prepare list, imm1`	`prepare list, 0`
	`mov imm1, r1`
	`sub r1, sp`

`prepare list, imm1, imm2`	`prepare list, 0, imm2`
	`mov imm1, r1`
	`sub r1, sp`

`dispose imm1, list, [reg1]`	`mov imm1, r1`
	`add r1, sp`
	`dispose 0, list, [reg1]`

`dispose imm1, list, [reg1]`	`mov imm1, r1`
	`add r1, sp`
	`dispose 0, list, [reg1]`

[Caution]

- When an absolute expression having a value exceeding the range of 0 to 127 is specified for imm1 in an instruction in the format of "prepare list, imm1, sp", the assembler outputs the following message and stops assembling.

  E0550231 : Illegal operand (range error in immediate).

- When a register that cannot be manipulated is specified for list, the assembler outputs the message shown below. Specification of that register is ignored in the generated code.

  W0550015 : Illegal register number, ignored.

- When an absolute expression having a value exceeding the range of 0 to 4095 is specified for list, the assembler outputs the following message and generates a code in which list is masked by 0xfff.

  W0550014 : Illegal list value, ignored.

- When an absolute expression that is not a multiple of 4 is specified for imm, the assembler outputs the following message and generates a code in which the lower two bits of imm are ignored.

  W0550019 : Illegal operand (immediate must be multiple of 4).

pushsp, popsp

- The pushsp and popsp instructions of the device should be specified in the formats of "pushsp rh-rt" and "popsp rh-rt", respectively.
  In asrh, they should be specified in the formats of "pushsp rh, rt" and "popsp rh, rt".

cmpf.d, cmpf.s

- The cmpf.d and cmpf.s instructions of the device should be specified in the format of "cmpf.* fcond, reg2, reg1, fcbit". In asrh, they can also be specified in the format of "cmpf*fcond*.* reg2, reg1, fcbit". Code that can be specified in cmpf*fcond* and the result of assembly are shown in Table 5.29.

Table 5.29 cmpf*fcond* Instruction List

Instruction	Condition	Meaning of Condition	Result of Assembly
cmpff.*	FALSE	Always false	cmpf.* 0x0
cmpfun.*	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.* 0x1
cmpfeq.*	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.* 0x2
cmpfueq.*	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.* 0x3
cmpfolt.*	reg2 < reg1	Neither is a non-number, and less than	cmpf.* 0x4
cmpfult.*	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.* 0x5
cmpfole.*	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.* 0x6
cmpfule.*	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.* 0x7
cmpfsf.*	FALSE	Always false	cmpf.* 0x8
cmpfngle.*	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.* 0x9
cmpfseq.*	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.* 0xA
cmpfngl.*	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.* 0xB
cmpflt.*	reg2 < reg1	Neither is a non-number, and less than	cmpf.* 0xC
cmpfnge.*	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.* 0xD
cmpfle.*	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.* 0xE
cmpfngt.*	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.* 0xF

Note ?: Unordered

[Example]

```
cmpfeq.s r10, r11, 0                    ; Same as "cmpf.s 0x2, r10, r11, 0"
```

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.

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 application. In C language, they can be specified using a #pragma section directive and in assembly language they can be specified using section definition directives.

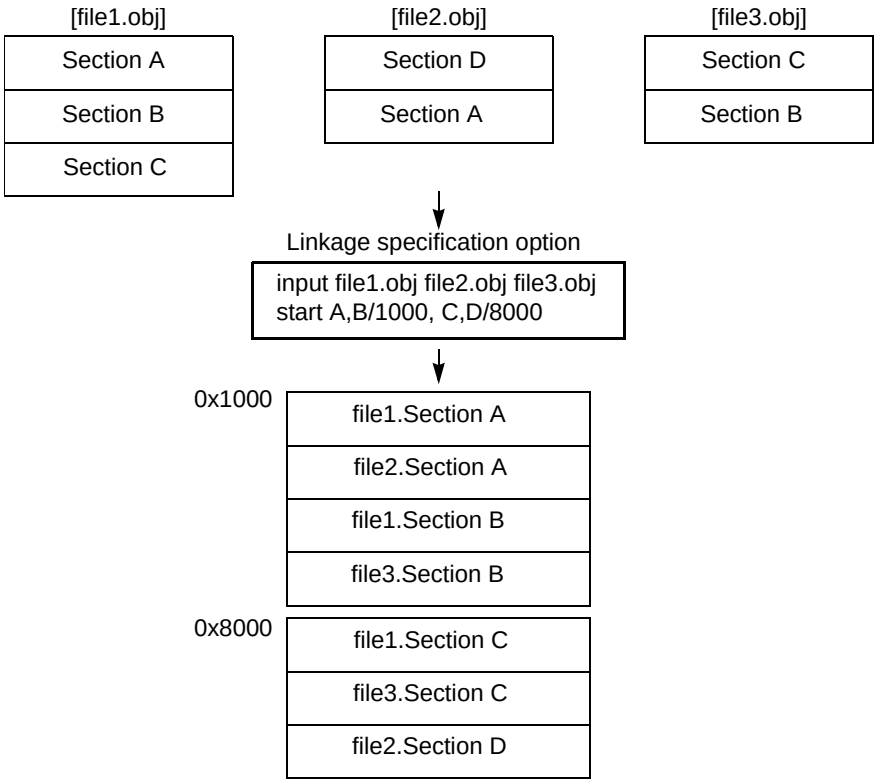
Even if the #pragma directive is not used to specify a section, however, allocation by the compiler to a particular section may already be set as the default setting in the program code or data (variables).

6.1.1 Section concatenation

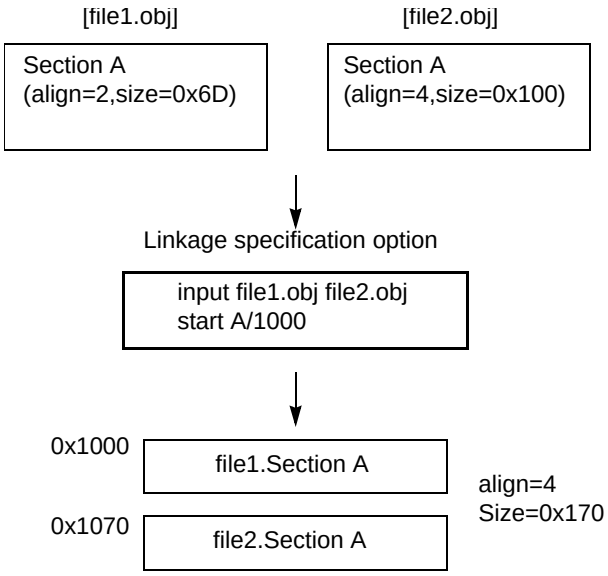
The optimizing linker (hereafter abbreviated "rlink") concatenates identical sections in the input relocatable files, and allocates them to the address specified by the -start option.

(1) Section allocation via the -start option

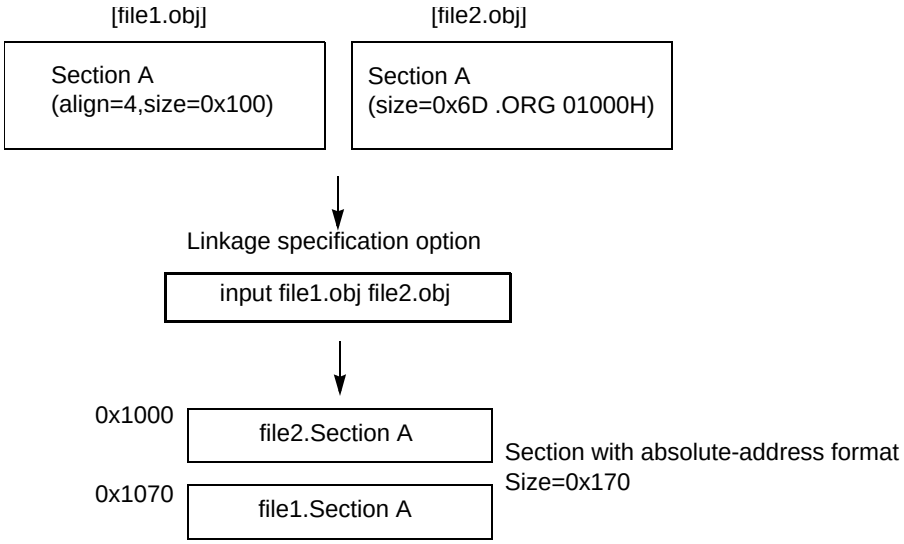
(a) Sections in different files with the same name are concatenated and allocated in the order of file input.



(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.

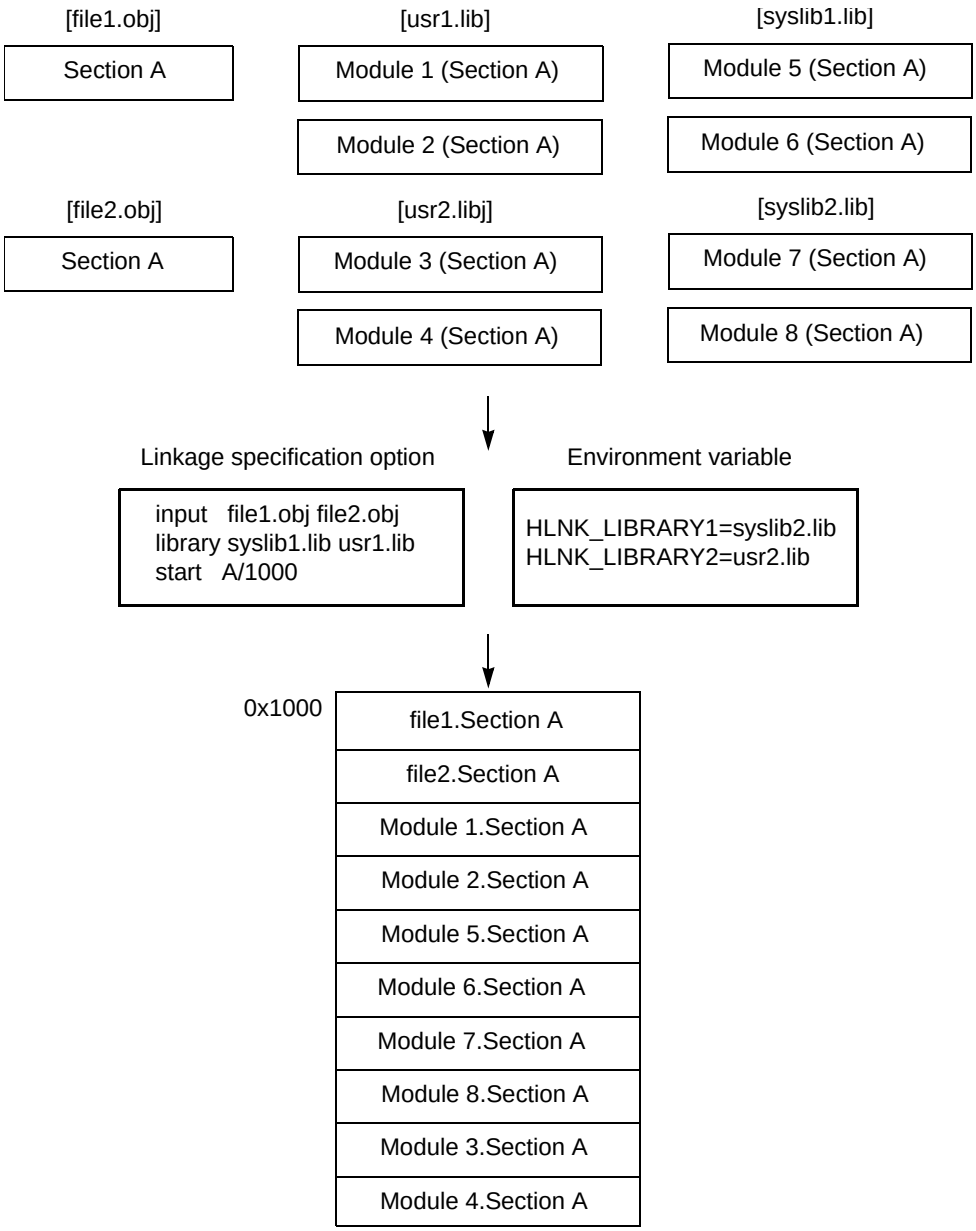


(c) If sections with the same name include both absolute-address format and relative-address format, then the sections with relative-address format are concatenated after the sections with absolute-address format.



(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



6.2 Special Symbol

The optimizing linker generates reserved symbols set to the values of the start addresses of each output section, and the first address beyond the end of each output section. If the user defines a symbol with the same name as one of these reserved symbols, then the optimizing linker will use the defined symbol, and will not generate its own.

The following two types of symbols are used as the name of the reserved symbol with the value of the start address of a section.

- Symbol with the string "__s" preceding the name of the output section
- Symbol with characters "@" and "." in the name of the output section replaced with "_" and the string "__S" added to the beginning [V1.06.00 or later]

The following two types of symbols are used as the name of the reserved symbol with the value of the first address after the end of a section.

- Symbol with the string "__e" preceding the name of the output section
- Symbol with characters "@" and "." in the name of the output section replaced with "_" and the string "__E" added to the beginning [V1.06.00 or later]

For example, when there are two sections, the .text section and FOO.const section, in an executable file, the eight special symbols shown below are generated.

__s.text
__S_text
__e.text
__E_text
__sFOO.const
__SFOO_const
__eFOO.const
__EFOO_const

Four of these special symbols (__S_text, __E_text, __SFOO_const, and __EFOO_const) can be referenced from the C source. To do this, specify as follows:

- Remove one underscore (_) from the beginning of the symbol name, and then declare the symbol as a char-type variable or char array-type variable with the extern specifier.

Example

```
extern char _S_text;
char* get_S_text(void) {
  return &_S_text;
}

extern char _E_text[];
char* get_E_text(void) {
  return _E_text;
}
```

7. LIBRARY FUNCTIONAL SPECIFICATIONS

This chapter describes the library functions provided in the CC-RH.

7.1 Supplied Libraries

The CC-RH provides the following libraries.

Table 7.1 Supplied Libraries
[V1.02.00 or later]

Supplied Libraries	Library Name	Conditions of Use
Standard library (Program diagnostic functions Functions with variable arguments Character string functions Memory management functions Character conversion functions Character classification functions Standard I/O functions Standard utility functions Peripheral device initialization function RAM section initialization function Operation runtime functions Checks for indirect function calls function) Mathematical library (Mathematical functions (Double-precision and single-precision))	lib\v850e3v5\rhf8n.lib	When -Xfloat=fpu, -Xdbl_size=8, and -Xround=nearest are specified
	lib\v850e3v5\rhs8n.lib	When -Xfloat=soft, -Xdbl_size=8, and -Xround=nearest are specified
	lib\v850e3v5\rhf4n.lib	When -Xfloat=fpu, -Xdbl_size=4, and -Xround=nearest are specified
	lib\v850e3v5\rhs4n.lib	When -Xfloat=soft, -Xdbl_size=4, and -Xround=nearest are specified
	lib\v850e3v5\rhf8z.lib	When -Xfloat=fpu, -Xdbl_size=8, and -Xround=zero are specified
	lib\v850e3v5\rhf4z.lib	When -Xfloat=fpu, -Xdbl_size=4, and -Xround=zero are specified
Standard library (Dynamic memory management functions) [V1.04.00 or later]	lib\v850e3v5\libmalloc.lib	Can always be used
	lib\v850e3v5\secure\libmalloc.lib	When the security facility is used [Professional Edition only]
Standard library (Non-local jump functions)	lib\v850e3v5\libsetjmp.lib	When -Xep=callee is specified
	lib\v850e3v5\ep\fix\libsetjmp.lib	When -Xep=fix is specified
Standard library (Program diagnostic functions Functions with variable arguments Character string functions Memory management functions Character conversion functions Character classification functions Standard I/O functions Standard utility functions Peripheral device initialization function RAM section initialization function Operation runtime functions)	lib\v850e3v5\libc.lib	Can always be used (for backward compatibility with V1.01.00 or earlier)
Mathematical library using FPU (Mathematical functions (Double-precision))	lib\v850e3v5\libm.lib	When -Xfloat=fpu is specified (for backward compatibility with V1.01.00 or earlier)
Mathematical library using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\libmf.lib	

Supplied Libraries	Library Name	Conditions of Use
Mathematical library not using FPU (Mathematical functions (Double-precision))	lib\v850e3v5\softfloat\libm.lib	When -Xfloat=soft is specified (for backward compatibility with V1.01.00 or earlier)
Mathematical library not using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\softfloat\libmf.lib	

[V1.01.00 or earlier]

Supplied Libraries	Library Name	Conditions of Use
Standard library (Program diagnostic functions Functions with variable arguments Character string functions Memory management functions Character conversion functions Character classification functions Standard I/O functions Standard utility functions Peripheral device initialization function RAM section initialization function Operation runtime functions)	lib\v850e3v5\libc.lib	Can always be used
Mathematical library using FPU (Mathematical functions (Double-precision))	lib\v850e3v5\libm.lib	When -Xfloat=fpu is specified
Mathematical library using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\libmf.lib	
Mathematical library not using FPU (Mathematical functions (Double-precision))	lib\v850e3v5\softfloat\libm.lib	When -Xfloat=soft is specified
Mathematical library not using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\softfloat\libmf.lib	
Standard library (Non-local jump functions)	lib\v850e3v5\libsetjmp.lib	When -Xep=callee is specified
	lib\v850e3v5\ep\fix\libsetjmp.lib	When -Xep=fix is specified

Table 7.2 Supplied Libraries (for -Xcpu=g3kh)

Supplied Libraries	Library Name	Conditions of Use
Standard library (Program diagnostic functions Functions with variable arguments Character string functions Memory management functions Character conversion functions Character classification functions Standard I/O functions Standard utility functions Peripheral device initialization function RAM section initialization function Operation runtime functions Checks for indirect function calls function) Mathematical library (Mathematical functions (Double-precision and single-precision))	lib \ v850e3v5 \ rhs8n.lib	When -Xfloat=fpu, -Xdbl_size=8, and -Xround=nearest are specified
	lib\v850e3v5\rhs8n.lib	When -Xdbl_size=8 and -Xround=nearest are specified
	lib\v850e3v5\rhf4n.lib	When -Xfloat=fpu, -Xdbl_size=4, and -Xround=nearest are specified
	lib\v850e3v5\rhs4n.lib	When -Xfloat=soft, -Xdbl_size=4, and -Xround=nearest are specified
	lib \ v850e3v5 \ rhs8n.lib	When -Xfloat=fpu, -Xdbl_size=8, and -Xround=zero are specified
	lib\v850e3v5\rhf4z.lib	When -Xfloat=fpu, -Xdbl_size=4, and -Xround=zero are specified
Standard library (Dynamic memory management functions) [V1.04.00 or later]	lib\v850e3v5\libmalloc.lib	Can always be used
	lib\v850e3v5\secure\libmalloc.lib	When the security facility is used [Professional Edition only]
Standard library (Non-local jump functions)	lib\v850e3v5\libsetjmp.lib	When -Xep=callee is specified
	lib\v850e3v5\ep\fix\libsetjmp.lib	When -Xep=fix is specified
Standard library (Program diagnostic functions Functions with variable arguments Character string functions Memory management functions Character conversion functions Character classification functions Standard I/O functions Standard utility functions Peripheral device initialization function RAM section initialization function Operation runtime functions Checks for indirect function calls function)	lib\v850e3v5\libc.lib	Can always be used (for backward compatibility with V1.01.00 or earlier)
Mathematical library (Mathematical functions (Double-precision))	lib\v850e3v5\softfloat\libm.lib	Can always be used (for backward compatibility with V1.01.00 or earlier)
Mathematical library using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\libmf.lib	When -Xfloat=fpu is specified (for backward compatibility with V1.01.00 or earlier)
Mathematical library not using FPU (Mathematical functions (Single-precision))	lib\v850e3v5\softfloat\libmf.lib	When -Xfloat=soft is specified (for backward compatibility with V1.01.00 or earlier)

- When the standard library or mathematical library is used in an application, include the related header files to use the library function.
  Refer these libraries using the optimizing linker option (-l).

However, it is not necessary to refer the libraries if only "program diagnosis function", "function with a variable arguments", "character conversion functions" and "character classification functions" are used.

- When CS+ is used, these libraries are referred by default.
- The operation runtime function is a routine that is automatically called by the CC-RH when a floating-point operation or integer operation is performed.
  Unlike the other library functions, the "operation runtime function" is not described in the C source or assembler source.
- Each rh***.lib library includes the contents of libc.lib, libm.lib, and libmf.lib (but does not include libmalloc.lib, libsetjmp.lib). When specifying an rh***.lib library, do not specify libc.lib, libm.lib, or libmf.lib together. In addition, only one of the rh***.lib libraries should be specified. Otherwise, correct operation is not guaranteed.
- secure\libmalloc.lib will generate an error if the Professional Edition license is not available.
- When linking libmalloc.lib, the standard library (rh***.lib) has to be linked at the same time.
- When the library (libc.lib) for achieving compatibility with earlier versions is used, the library for the security facility cannot be used.

7.2 Header Files

The list of header files required for using the libraries of the CC-RH are listed below.
The macro definitions and function declarations are described in each file.

Table 7.3 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
half.h [V1.05.00 or later]	Header file for half-precision floating-point representation
limits.h	Header file for quantitative limiting of integers
math.h	Header file for mathematical calculation
mathf.h	Header file for mathematical calculation (declares single-precision math functions and defines single-precision macros outside of the C90 standard)
setjmp.h	Header file for non-local jump
stdarg.h	Header file for supporting functions having variable arguments
stddef.h	Header file for common definitions
stdio.h	Header file for standard I/O
stdlib.h	Header file for standard utilities
string.h	Header file for memory manipulation and character string manipulation
iso646.h [V1.07.00 or later]	Header file for alternative spelling macro
stdbool.h [V1.07.00 or later]	Header file for boolean type and values
stdint.h [V1.07.00 or later]	Header file for integer type of specified width
_h_c_lib.h	Header file for the initial setting routine

7.3 Reentrancy

"Reentrancy (re-enter is possible)" means that a certain process can be called concurrently at the same time from multiple processes. A reentrancy function can be correctly executed in another process while the function is being executed. For example, in an application using a real-time OS, this function is correctly executed even if dispatching to another task is triggered by an interrupt while a certain task is executing this function, and even if the function is executed in that task. When data variables or RAM are shared by multiple functions, those functions may not necessarily be reentrant.

7.4 Library Function

This section explains Library Function.

7.4.1 Program diagnostic functions

Program diagnostic functions are as follows

Table 7.4 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>
assert(int *expression*);

[Description]

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[Note].

The diagnostic information includes the program text of the parameters, the name of the source file, and the line number of the source.

If you wish to disable the assert macro, include a #define NDEBUG statement before assert.h is loaded.

Note If you use the assert macro, you must create an abort function in accordance with the user system.

[Example]

```
#include    <assert.h>
int func(void);
int main() {
    int ret;
    ret = func();
    assert(ret == 0);       <- abort() is called if ret is not 0
    return 0;
}
```

7.4.2 Functions with variable arguments

Functions with a variable arguments are as follows

Table 7.5 Functions with Variable Arguments

Function/Macro Name	Outline
va_start	Initialization of variable for scanning argument list
va_end	End of scanning argument list
va_arg	Moving variable for scanning argument list

va_start

Initialization of variable for scanning argument list

[Classification]

Standard library

[Syntax]

#include <stdarg.h>
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.

[Example]

```
#include    <stdarg.h>
void abc(int first, int second, ...) {
        va_list ap;
        int     i;
        char    c, *fmt;
        va_start(ap, second);
        i = va_arg(ap, int);
        c = va_arg(ap, int);        /*char type is converted into int type.*/
        fmt = va_arg(ap, char *);
        va_end(ap);
}
```

va_end

End of scanning argument list

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
void    va_end(va_list ap);
```

[Description]

This function indicates the end of scanning the list. By enclosing va_arg between va_start and va_end, scanning the list can be repeated.

va_arg

Moving variable for scanning argument list

[Classification]

Standard library

[Syntax]

```
#include <stdarg.h>
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. With the C compiler specify the int type for an argument of char and short types, and specify the unsigned int type for an argument of unsigned char and unsigned short types. 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.

Also stipulate "how many functions are actually passed" according to the conventions between the called function and calling function.

7.4.3 Character string functions

Character string functions are as follows.

Table 7.6 Character String Functions

Function/Macro Name	Outline
strpbrk	Character string search (start position)
strrchr	Character string search (end position)
strchr	Character string search (start position of specified character)
strstr	Character string search (start position of specified character string)
strspn	Character string search (maximum length including specified character)
strcspn	Character string search (maximum length not including specified character)
strcmp	Character string comparison
strncmp	Character string comparison (with number of characters specified)
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)
strtok	Token division
strlen	Length of character string
strerror	Character string conversion of error number

strpbrk

Character string search (start position)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strpbrk(const char *s1, const char *s2);
```

[Return value]

Returns the pointer indicating this character. 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 string search (end position)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strrchr(const char *s, int c);
```

[Return value]

Returns a pointer indicating *c* 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* converted into char type appears last in the character string indicated by *s*. The null character (\0) indicating termination is regarded as part of this character string.

strchr

Character string search (start position of specified character)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strchr(const char *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* converted into char type appears in the character string indicated by *s*. The null character (\0) indicating termination is regarded as part of this character string.

strstr

Character string search (start position of specified character)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strstr(const char *s1, const char *s2);
```

[Return value]

Returns the pointer indicating the character string that has been found. If character string *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*.

strspn

Character string search (maximum length including specified character)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t  strspn(const char *s1, const char *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*.

strcspn

Character string search (maximum length not including specified character)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t  strcspn(const char *s1, const char *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*.

strcmp

Character string comparison

[Classification]

Standard library

[Syntax]

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

[Return value]

Returns an integer 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 strncmp(const char *s1, const char *s2, size_t length);
```

[Return value]

Returns an integer 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 up to *length* characters of the array indicated by *s1* with characters of the array indicated by *s2*.

strcpy

Character string copy

[Classification]

Standard library

[Syntax]

#include <string.h>
char *strcpy(char **dst*, const char **src*);

[Return value]

Returns the value of *dst*.

[Description]

This function copies the character string indicated by *src* to the array indicated by *dst*.

[Example]

```
#include    <string.h>
void func(char *str, const char *src) {
        strcpy(str, src);   /*Copies character string indicated by src to array
                              indicated by str.*/
          :
}
```

strncpy

Character string copy (with number of characters specified)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strncpy(char *dst, const char *src, size_t length);
```

[Return value]

Returns the value of *dst*.

[Description]

This function copies up to *length* characters (including the null character (\0)) from the array indicated by *src* to the array indicated by *dst*. If the array indicate by *src* is shorter than *length* characters, null characters (\0) are appended to the duplication in the array indicated by *dst*, until all *length* characters are written.

strcat

Character string concatenation

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strcat(char *dst, const char *src);
```

[Return value]

Returns the value of *dst*.

[Description]

This function concatenates the duplication of the character string indicated by *src* to the end of the character string indicated by *dst*, including the null character (\0). The first character of *src* overwrites the null character (\0) at the end of *dst*.

strncat

Character string concatenation (with number of characters specified)

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strncat(char *dst, const char *src, size_t length);
```

[Return value]

Returns the value of *dst*.

[Description]

This function concatenates up to *length* characters (including the null character (\0) of *src*) to the end of the character string indicated by *dst*, starting from the beginning of the character string indicated by *src*. The null character (\0) at the end of *dst* is written over the first character of *src*. The null character indicating termination (\0) is always added to this result.

[Caution]

Because the null character (\0) is always appended when strncat is used, if copying is limited by the number of *length* arguments, the number of characters appended to *dst* is *length* + 1.

strtok

Token division

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strtok(char *s, const char *delimiters);
```

[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 *s* into strings of tokens by delimiting the character string with a character in the character string indicated by *delimiters*. If this function is called first, *s* is used as the first argument. Then, calling with the null pointer as the first argument continues. The delimiting character string indicated by *delimiters* 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 *delimiters*. If such a character is not found, a token does not exist in the character string indicated by *s*, 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 *s*, 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.

strlen

Length of character string

[Classification]

Standard library

[Syntax]

```
#include <string.h>
size_t  strlen(const char *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*.

strerror

Character string conversion of error number

[Classification]

Standard library

[Syntax]

```
#include <string.h>
char    *strerror(int errnum);
```

[Return value]

Returns a pointer to the converted character string.

[Description]

This function converts error number *errnum* into a character string. The value of *errnum* is usually the duplication of global variable errno. Do not change the specified array of the application program.

The message that is output is as follows.

When errno is EDOM	"EDOM error"
When errno is ERANGE	"ERANGE error"
When errno is 0	"no error"
Otherwise	"error *xxx*" (*xxx* is abs (errno) % 1000)

7.4.4 Memory management functions

Memory management functions are as follows.

Table 7.7 Memory Management Functions

Function/Macro Name	Outline
memchr	Memory search
memcmp	Memory comparison
memcpy	Memory copy
memmove	Memory move
memset	Memory set

memchr

Memory search

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void    *memchr(const void *s, int c, size_t length);
```

[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* (converted into char type) appears first in the first *length* number of characters in an area indicated by *s*.

memcmp

Memory comparison

[Classification]

Standard library

[Syntax]

#include <string.h>
int memcmp(const void *s1, const void *s2, size_t n);

[Return value]

An integer 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*.

[Example]

```
#include    <string.h>
int func(const void *s1, const void *s2) {
        int i;
        i = memcmp(s1, s2, 5);  /*Compares the first five characters of the character
                                  string indicated by s1 with the first five
                                  characters of the character string indicated by
                                  s2.*/
        return(i);
}
```

memcpy

Memory copy

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void    *memcpy(void *out, const void *in, size_t n);
```

[Return value]

Returns the value of *out*. The operation is undefined if the copy source and copy destination areas overlap.

[Description]

This function copies *n* bytes from an object indicated by *in* to an object indicated by *out*.

memmove

Memory move

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void    *memmove(void *dst, void *src, size_t length);
```

[Return value]

Returns the value of *dst* at the copy destination.

[Description]

This function moves the *length* number of characters from a memory area indicated by *src* to a memory area indicated by *dst*. Even if the copy source and copy destination areas overlap, the characters are correctly copied to the memory area indicated by *dst*.

memset

Memory set

[Classification]

Standard library

[Syntax]

```
#include <string.h>
void    *memset(const void *s, int c, size_t length);
```

[Return value]

Returns the value of *s*.

[Description]

This function copies the value of *c* (converted into unsigned char type) to the first *length* character of an object indicated by *s*.

7.4.5 Character conversion functions

Character conversion functions are as follows.

Table 7.8 Character Conversion Functions

Function/Macro Name	Outline
toupper	Conversion from lower-case to upper-case (not converted if argument is not in lower-case)
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)

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int 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.

This macro is defined only when *c* is an integer in the range of EOF to 255. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef toupper".

[Example]

```
#include    <ctype.h>
int c = 'a';
int func() {
        int i;
        i = toupper(c);         /*Converts lowercase character 'a' of c into uppercase
                                  character 'A'.*/
        return(i);
}
```

tolower

Conversion from upper-case to lower-case (not converted if argument is not in upper-case)

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int 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.

This macro is defined only when *c* is an integer in the range of EOF to 255. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef tolower".

7.4.6 Character classification functions

Character classification functions are as follows.

Table 7.9 Character Classification Functions

Function/Macro Name	Outline
isalnum	Identification of ASCII letter or numeral
isalpha	Identification of ASCII letter
isascii	Identification of ASCII code
isupper	Identification of upper-case character
islower	Identification of lower-case character
isdigit	Identification of decimal number
isxdigit	Identification of hexadecimal number
iscntrl	Identification of control character
ispunct	Identification of delimiter character
isspace	Identification of space/tab/carriage return/line feed/vertical tab/page feed
isprint	Identification of display character
isgraph	Identification of display character other than space

isalnum

Identification of ASCII letter or numeral

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isalnum(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is an ASCII alphabetic character or numeral. This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isalnum".

isalpha

Identification of ASCII letter

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isalpha(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is an ASCII alphabetic character. This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isalpha".

isascii

Identification of ASCII code

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isascii(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is an ASCII code (0x00 to 0x7F). This macro is defined for all integer values. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isascii".

isupper

Identification of upper-case character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isupper(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is an uppercase character (A to Z). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isupper".

islower

Identification of lower-case character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int islower(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a lowercase character (a to z). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef islower".

isdigit

Identification of decimal number

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isdigit(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a decimal number. This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isdigit".

isxdigit

Identification of hexadecimal number

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isxdigit(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a hexadecimal number (0 to 9, a to f, or A to F). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isxdigit".

iscntrl

Identification of control character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int iscntrl(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a control character (0x00 to 0x1F or 0x7F). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef iscntrl".

ispunct

Identification of delimiter character

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int ispunct(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a printable delimiter (isgraph(c) && !isalnum(c)). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef ispunct".

isspace

Identification of space/tab/carriage return/line feed/vertical tab/page feed

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isspace(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a space, tap, line feed, carriage return, vertical tab, or form feed (0x09 to 0x0D, or 0x20). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isspace".

isprint

Identification of display character

[Classification]

Standard library

[Syntax]

#include <ctype.h>
int isprint(int *c*);

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a display character (0x20 to 0x7E). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isprint".

[Example]

```
#include    <ctype.h>
void func(void) {
        int     i, j = 0;
        char    s[50];
                for (i =50; i <= 99; i++) {
                        if (isprint(i)) {   /*Store the printable characters in the
                                              code range 50 to 99, in the array s.*/
                        s[j] = i;
                        j++;
                        }
                }
        :
}
```

isgraph

Identification of display character other than space

[Classification]

Standard library

[Syntax]

```
#include <ctype.h>
int isgraph(int c);
```

[Return value]

These macros return 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.

[Description]

This function is a macro that checks whether a given character is a display character[Note] (0x20 to 0x7E) other than space (0x20). This macro is defined only when *c* is made true by isascii or when *c* is EOF. A compiled subroutine can be used instead of the macro definition, which is invalidated by using "#undef isgraph".

Note printing character

7.4.7 Standard I/O functions

Standard I/O functions are as follows.

Table 7.10 Standard I/O Functions

Function/Macro Name	Outline
fread	Read from stream
getc	Read character from stream (same as fgetc)
fgetc	Read character from stream (same as getc)
fgets	Read one line from stream
fwrite	Write to stream
putc	Write character to stream (same as fputc)
fputc	Write character to stream (same as putc)
fputs	Output character string to stream
getchar	Read one character from standard input
gets	Read character string from standard input
putchar	Write character to standard output stream
puts	Output character string to standard output stream
sprintf	Output with format
fprintf	Output text in specified format to stream
vsprintf	Write text in specified format to character string
printf	Output text in specified format to standard output stream
vfprintf	Write text in specified format to stream
vprintf	Write text in specified format to standard output stream
sscanf	Input with format
fscanf	Read and interpret data from stream
scanf	Read and interpret text from standard input stream
ungetc	Push character back to input stream
rewind	Reset file position indicator
perror	Error processing

fread

Read from stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
size_t fread(void *_ptr_, size_t _size_, size_t _nmemb_, FILE *_stream_);

[Return value]

The number of elements that were input (_nmemb_) is returned.
Error return does not occur.

[Description]

This function inputs _nmemb_ elements of _size_ from the input stream pointed to by _stream_ and stores them in _ptr_. Only the standard input/output stdin can be specified for _stream_.

[Example]

```
#include    <stdio.h>
void func(void) {
        struct {
                int     c;
                double  d;
        } buf[10];
        fread(buf, sizeof(buf[0]), sizeof(buf) / sizeof(buf [0]), stdin);
}
```

getc

Read character from stream (same as fgetc)

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int getc(FILE *stream);
```

[Return value]

The input character is returned.
Error return does not occur.

[Description]

This function inputs one character from the input stream pointed to by *stream*. Only the standard input/output stdin can be specified for *stream*.

fgetc

Read character from stream (same as getc)

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int fgetc(FILE *_stream_);

[Return value]

The input character is returned.
Error return does not occur.

[Description]

This function inputs one character from the input stream pointed to by *stream*. Only the standard input/output stdin can be specified for *stream*.

[Example]

```
#include    <stdio.h>

int func(void) {
        int c;
        c = fgetc(stdin);
        return(c);
}
```

fgets

Read one line from stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
char    *fgets(char *s, int n, FILE *stream);
```

[Return value]

s is returned.
Error return does not occur.

[Description]

This function inputs at most *n-1* characters from the input stream pointed to by *stream* and stores them in *s*. Character input is also ended by the detection of a new-line character. In this case, the new-line character is also stored in *s*. The end-of-string null character is stored at the end in *s*. Only the standard input/output stdin can be specified for *stream*.

fwrite

Write to stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
size_t  fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[Return value]

The number of elements that were output (*nmemb*) is returned.
Error return does not occur.

[Description]

This function outputs *nmemb* elements of *size* from the array pointed to by *ptr* to the output stream pointed to by *stream*. Only the standard input/output stdout or stderr can be specified for *stream*.

putc

Write character to stream (same as fputc)

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

[Return value]

The character *c* is returned.
Error return does not occur.

[Description]

This function outputs the character *c* to the output stream pointed to by *stream.* Only the standard input/output stdout or stderr can be specified for *stream*.

fputc

Write character to stream (same as putc)

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int fputc(int *c*, FILE **stream*);

[Return value]

The character *c* is returned.
Error return does not occur.

[Description]

This functionoutputs the character *c* to the output stream pointed to by *stream*. Only the standard input/output stdout or stderr can be specified for *stream*.

[Example]

```
#include    <stdio.h>
void func(void) {
        fputc('a', stdout);
}
```

fputs

Output character string to stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

[Return value]

0 is returned.
Error return does not occur.

[Description]

This function outputs the string *s* to the output stream pointed to by *stream*. The end-of-string null character is not output. Only the standard input/output stdout or stderr can be specified for *stream*.

getchar

Read one character from standard input

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int getchar(void);
```

[Return value]

The input character is returned.
Error return does not occur.

[Description]

This function inputs one character from the standard input/output stdin.

gets

Read character string from standard input

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
char    *gets(char *s);
```

[Return value]

s is returned.
Error return does not occur.

[Description]

This function inputs characters from the standard input/output stdin until a new-line character is detected and stores them in *s*. The new-line character that was input is discarded, and an end-of-string null character is stored at the end in *s*.

putchar

Write character to standard output stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int putchar(int c);
```

[Return value]

The character *c* is returned.
Error return does not occur.

[Description]

This function outputs the character *c* to the standard input/output stdout.

puts

Output character string to standard output stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int puts(const char *s);
```

[Return value]

0 is returned.
Error return does not occur.

[Description]

This function outputs the string *s* to the standard input/output stdout. The end-of-string null character is not output, but a new-line character is output in its place.

sprintf

Output with format

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int sprintf(char *s, const char *format[, arg, ...]);
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Error return does not occur.

[Description]

This function applies the format specified by the string pointed to by *format* to the respective *arg* arguments, and writes out the formatted data that was output as a result to the array pointed to by *s*.

If there are not sufficient arguments for the format, the operation is undefined. If the end of the formatted string is reached, control returns. If there are more arguments that those required by the format, the excess arguments are ignored. If the area of *s* overlaps one of the arguments, the operation is undefined.

The argument *format* specifies "the output to which the subsequent argument is to be converted". The null character (\0) is appended at the end of written characters (the null character (\0) is not counted in a return value).

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 to an alternate 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, 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, 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, or E 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. 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 h, l, ll, or L, which changes the default method for interpreting the data type of the corresponding argument.
When h is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a short or unsigned 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. If another type specification character is used together with h or l, the operation is undefined.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly applied to a long long and unsigned long long argument. Furthermore, for ll, a following n type specification is forcibly applied to a long long pointer. If another type specification character is used together with ll, the operation is undefined.
When L is specified, a following e, E, f, g, or G type specification is forcibly applied to a long double argument. If another type specification character is used together with L, the operation is undefined.

(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.

%	Output the character "%". No argument is converted. The conversion specification is "%%".
c	Convert an int type argument to unsigned char type and output the characters of the conversion result.
d	Convert an int type argument to a signed decimal number.
e, E	Convert a double type 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".
f	Convert a double type argument to decimal notation of the form [-]dddd.dddd.

g, G	Convert a double type 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.
i	Perform the same conversion as d.
n	Store the number of characters that were output in the same object. A pointer to int type is used as the argument.
p	Output a pointer value. The CC-RH handles a pointer as unsigned long (this is the same as the lu specification).
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.
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).

[Example]

```
#include    <stdio.h>
void func(int val) {
        char    s[20];
       sprintf(s, "%-10.51x\n", val);  /*Specifies left-justification, field width 10,
                                          precision 5, size long, and hexadecimal
                                          notation for the value of val, and outputs
                                          the result with an appended new-line
                                          character to the array pointed to by s.*/
}
```

fprintf

Output text in specified format to stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int fprintf(FILE *_stream_, const char *_format_[, _arg_, ...]);

[Return value]

The number of characters that were output is returned.

[Description]

This function applies the format specified by the string pointed to by *format* to the respective *arg* arguments, and outputs the formatted data that was output as a result to *stream*. Only the standard input/output stdout or stderr can be specified for *stream*. The method of specifying *format* is the same as described for the sprintf function. However, fprintf differs from sprintf in that no null character (\0) is output at the end.

[Caution]

Stdin (standard input) and stdout (standard error) are specified for the argument *stream*. 1 memory addresses such as an I/O address is allocated for the I/O destination of stream. To use these streams in combination with a debugger, the initial values of the stream structure defined in stdio.h must be set. Be sure to set the initial values prior to calling the function.

[Definition of stream structure in stdio.h]

```
typedef struct {
        int         mode;   /*with error descriptions*/
        unsigend    handle;
        int         unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())
#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

The first structure member, mode, indicates the I/O status and is internally defined as ACCSD_OUT/ADDSD_IN. The third member, unget_c, indicates the pushed-back character (stdin only) setting and is internally defined as -1.

When the definition is -1, it indicates that there is no pushed-back character. The second member, handle, indicates the I/O address. Set the value according to the debugger to be used.

[I/O address setting]

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

[Example]

```
#include    <stdio.h>
void func(int val) {
        fprintf(stdout, "%-10.5x\n", val);
}
/*Example using vfprintf in a general error reporting routine.*/
void error(char *function_name, char *format, ...) {
        va_list arg;
        va_start(arg, format);
        fprintf(stderr, "ERROR in %s:", function_name); /*output function name for
                                                          which error occurred*/
        vfprintf(stderr, format, arg);                 /*output remaining messages*/
        va_end(arg);
}
```

vsprintf

Write text in specified format to character string

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int vsprintf(char *s, const char *format, va_list arg);
```

[Return value]

The number of characters that were output (excluding the null character (\0)) is returned.
Error return does not occur.

[Description]

This function applies the format specified by the string pointed to by *format* to the argument string pointed to by *arg*, and outputs the formatted data that was output as a result to the array pointed to be *s*. The vsprintf function is equivalent to sprintf with the list of a variable number of real arguments replaced by *arg*. *arg* must be initialized by the va_start macro before the vsprintf function is called.

printf

Output text in specified format to standard output stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int printf(const char *_format_[, _arg_, ...]);

[Return value]

The number of characters that were output is returned.

[Description]

This function applies the format specified by the string pointed to by *format* to the respective *arg* arguments, and outputs the formatted data that was output as a result to the standard input/output stdout. The method of specifying *format* is the same as described for the sprintf function. However, printf differs from sprintf in that no null character (\0) is output at the end.

[Caution]

Assigns one memory address (e.g. an I/O address) to stdout. To use stdout in conjunction with a debugger, you must initialize the stream structure defined in the stdio.h file. Initialize the structure before calling the function.

[Definition of stream structure in stdio.h]

```
typedef struct {
        int         mode;   /*with error descriptions*/
        unsigend    handle;
        int         unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())

#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

The first structure member, mode, indicates the I/O status and is internally defined as ACCSD_OUT/ADDSD_IN. The third member, unget_c, indicates the pushed-back character (stdin only) setting and is internally defined as -1.

When the definition is -1, it indicates that there is no pushed-back character. The second member, handle, indicates the I/O address. Set the value according to the debugger to be used.

[I/O address setting]

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

vfprintf

Write text in specified format to stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int vfprintf(FILE **stream*, const char **format*, va_list *arg*);

[Return value]

The number of characters that were output is returned.

[Description]

This function applies the format specified by the string pointed to by *format* to argument string pointed to by *arg*, and outputs the formatted data that was output as a result to *stream*. Only the standard input/output stdout or stderr can be specified for *stream*. The method of specifying *format* is the same as described for the sprintf function. The vfprintf function is equivalent to sprintf with the list of a variable number of real arguments replaced by *arg*. *arg* must be initialized by the va_start macro before the vfprintf function is called.

[Caution]

Stdout (standard output) and stderr (standard error) are specified for the argument *stream*. 1 memory addresses such as an I/O address is allocated for the I/O destination of stream. To use these streams in combination with a debugger, the initial values of the stream structure defined in stdio.h must be set. Be sure to set the initial values prior to calling the function.

[Definition of stream structure in stdio.h]

```
typedef struct {
        int         mode;   /*with error descriptions*/
        unsigend    handle;
        int         unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())

#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

The first structure member, mode, indicates the I/O status and is internally defined as ACCSD_OUT/ADDSD_IN. The third member, unget_c, indicates the pushed-back character (stdin only) setting and is internally defined as -1.

When the definition is -1, it indicates that there is no pushed-back character. The second member, handle, indicates the I/O address. Set the value according to the debugger to be used.

[I/O address setting]

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

[Example]

```
#include    <stdio.h>
void func(int val) {
        fprintf(stdout, "%-10.5x\n", val);
}
/*example using vfprintf in a general error reporting routine*/
void error(char *function_name, char *format, ...) {
        va_list arg;
        va_start(arg, format);
        fprintf(stderr, "ERROR in %s:", function_name); /*output function name for
                                                          which error occurred*/
        vfprintf(stderr, format, arg);                 /*output remaining messages*/
        va_end(arg);
}
```

vprintf

Write text in specified format to standard output stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int vprintf(const char *format, va_list arg);
```

[Return value]

The number of characters that were output is returned.

[Description]

This function applies the format specified by the string pointed to by *format* to the argument string pointed to by *arg*, and outputs the formatted data that was output as a result to the standard input/output stdout. The method of specifying *format* is the same as described for the sprintf function. The vprintf function is equivalent to sprintf with the list of a variable number of real arguments replaced by *arg*. *arg* must be initialized by the va_start macro before the vprintf function is called.

[Caution]

Stdout (standard output) and stderr (standard error) are specified for the argument *stream*. 1 memory addresses such as an I/O address is allocated for the I/O destination of stream. To use these streams in combination with a debugger, the initial values of the stream structure defined in stdio.h must be set. Be sure to set the initial values prior to calling the function.

[Definition of stream structure in stdio.h]

```
typedef struct {
        int         mode;   /*with error descriptions*/
        unsigend    handle;
        int         unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())

#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

The first structure member, mode, indicates the I/O status and is internally defined as ACCSD_OUT/ADDSD_IN. The third member, unget_c, indicates the pushed-back character (stdin only) setting and is internally defined as -1.

When the definition is -1, it indicates that there is no pushed-back character. The second member, handle, indicates the I/O address. Set the value according to the debugger to be used.

[I/O address setting]

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

sscanf

nput with format

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int sscanf(const char *s, const char *format[, arg, ...]);
```

[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 reads the input to be converted according to the *format* specified by the character string pointed to by format from the array pointed to by *s* and treats the *arg* arguments that follow format as pointers that point to objects for storing the converted input.

An input string that can be recognized and "the conversion that is to be performed for assignment" are specified for *format*. If sufficient arguments do not exist for *format*, the operation is undefined. If *format* is used up even when arguments remain, the remaining arguments are ignored.

The *format* consists of the following three types of directives:

One or more Space characters	Space ( ), tab (\t), or new-line (\n). If a space character is found in the string when sscanf is executed, all consecutive space characters are read until the next non-space character appears (the space characters are not stored).
Ordinary characters	All ASCII characters other than "%". If an ordinary character is found in the string when sscanf is executed, that character is read but not stored. sscanf reads a string from the input field, converts it into a value of a specific type, and stores it at the position specified by the argument, according to the conversion specification. If an explicit match does not occur according to the conversion specification, no subsequent space character is read.
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 the interpretation and assignment of the input field.

(2) field width
This is a non-zero decimal integer that defines the maximum field width.
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 h, l, ll, or L, which changes the default method for interpreting the data type of the corresponding argument.
When h is specified, a following d, i, n, o, u, or x type specification is forcibly converted to short int type and stored as short type. Nothing is done for c, e, f, n, p, s, D, I, O, U, or X.
When l is specified, a following d, i, n, o, u, or x type specification is forcibly converted to long int type and stored as long type. An e, f, or g type specification is forcibly converted to double type and stored as double type. Nothing is done for c, n, p, s, D, I, O, U, and X.
When ll is specified, a following d, i, o, u, x, or X type specification is forcibly converted to long long type and stored as long long type. Nothing is done for other type specifications.
When L is specified, a following e, f, or g type specification is forcibly converted to long double type and stored as long double type. Nothing is done for other type specifications.
In cases other than the above, the operation is undefined.

(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.

%	Match the character "%". No conversion or assignment is performed. The conversion specification is "%%".
c	Scan one character. The corresponding argument should be "char **arg*".
d	Read a decimal integer into the corresponding argument. The corresponding argument should be "int **arg*".
e, f, g	Read a floating-point number into the corresponding argument. The corresponding argument should be "float **arg*".
i	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding argument should be "int **arg*".
n	Store the number of characters that were read in the corresponding argument. The corresponding argument should be "int **arg*".
o	Read an octal integer into the corresponding argument. The corresponding argument must be "int **arg*".
p	Store the pointer that was scanned. The ca processes %p and %U in exactly the same manner. The corresponding argument should be "void ***arg*".
s	Read a string into a given array. The corresponding argument should be "char *arg*[ ]".
u	Read an unsigned decimal integer into the corresponding argument. The corresponding argument should be "unsigned int **arg*".
x, X	Read a hexadecimal integer into the corresponding argument. The corresponding argument should be "int **arg*".
D	Read a decimal integer into the corresponding argument. The corresponding argument should be "long **arg*".
E, F, G	Read a floating-point number into the corresponding argument. The corresponding argument should be "double **arg*".
I	Read a decimal, octal, or hexadecimal integer into the corresponding argument. The corresponding argument should be "long **arg*".
O	Read an octal integer into the corresponding argument. The corresponding argument should be "long **arg*".
U	Read an unsigned decimal integer into the corresponding argument. The corresponding argument should be "unsigned long **arg*".

[ ]	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).

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.

[Example]

```
#include    <stdio.h>
void func(void) {
        int         i, n;
        float       x;
        const char  *s;
        char        name[10];
        s = "23 11.1e-1 NAME";
        n = sscanf(s,"%d%f%s", &i, &x, name);   /*Stores 23 in i, 1.110000 in x, and
                                                  "NAME" in name.  The return value n
                                                  is 3.*/
}
```

fscanf

Read and interpret data from stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format[, arg, ...]);
```

[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]

Reads the input to be converted according to the format specified by the character string pointed to by *format* from *stream* and treats the *arg* arguments that follow format as objects for storing the converted input. Only the standard input/output stdin can be specified for *stream*. The method of specifying *format* is the same as described for the sscanf function.

scanf

Read and interpret text from standard output stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

#include <stdio.h>
int scanf(const char **format*[, *arg*, ...]);

[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]

Reads the input to be converted according to the format specified by the character string pointed to by *format* from the standard input/output stdin and treats the *arg* arguments that follow format as objects for storing the converted input. The method of specifying *format* is the same as described for the sscanf function.

[Example]

```
#include    <stdio.h>
void func(void) {
        int     i, n;
        double  x;
        char    name[10];
        n = scanf("%d%lf%s", &i, &x, name); /*Perform formatted input of input from
                                               stdin using the format
                                               "23 11.1e-1 NAME".*/
}
```

ungetc

Push character back to input stream

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

[Return value]

The character *c* is returned.
Error return does not occur.

[Description]

This function pushes the character *c* back into the input stream pointed to by *stream*. However, if *c* is EOF, no pushback is performed. The character *c* that was pushed back will be input as the first character during the next character input. Only one character can be pushed back by ungetc. If ungetc is executed continuously, only the last ungetc will have an effect. Only the standard input/output stdin can be specified for *stream*.

rewind

Reset file position indicator

Remark These functions are not supported by the debugging functions which CS+ provides.

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
void    rewind(FILE *stream);
```

[Description]

This function clears the error indicator of the input stream pointed to by *stream*, and positions the file position indicator at the beginning of the file.

However, only the standard input/output stdin can be specified for *stream*. Therefore, rewind only has the effect of discarding the character that was pushed back by ungetc.

perror

Error processing

[Classification]

Standard library

[Syntax]

```
#include <stdio.h>
void    perror(const char *s);
```

[Description]

This function outputs to stderr the error message that corresponds to global variable errno.
The message that is output is as follows.

When *s* is not NULL	`fprintf(stderr, "%s:%s\n", s, s_fix);`
When *s* is NULL	`fprintf(stderr, "%s\n", s_fix);`

s_fix is as follows.

When errno is EDOM	"EDOM error"
When errno is ERANGE	"ERANGE error"
When errno is 0	"no error"
Otherwise	"error *xxx*" (*xxx* is abs (errno) % 1000)

[Example]

```
#include    <stdio.h>
#include    <errno.h>
void func(double x) {
        double  d;
        errno = 0;
        d = exp(x);
        if(errno)
                perror("func1");    /*If a calculation exception is generated by exp
                                      perror is called.*/
}
```

7.4.8 Standard utility functions

Standard Utility functions are as follows.

Table 7.11 Standard Utility Functions

Function/Macro Name	Outline
abs	Output absolute value (int type)
labs	Output absolute value (long type)
llabs	Output absolute value (long long type)
bsearch	Binary search
qsort	Sort
div	Division (int type)
ldiv	Division (long type)
lldiv	Division (long long type)
atoi	Conversion of character string to integer (int type)
atol	Conversion of character string to integer (long type)
atoll	Conversion of character string to integer (long long type)
strtol	Conversion of character string to integer (long type) and storing pointer in last character string
strtoul	Conversion of character string to integer (unsigned long type) and storing pointer in last character string
strtoll	Conversion of character string to integer (long long type) and storing pointer in last character string
strtoull	Conversion of character string to integer (unsigned long long type) and storing pointer in last character string
atoff	Conversion of character string to floating-point number (float type)
atof	Conversion of character string to floating-point number (double type)
strtodf	Conversion of character string to floating-point number (float type) (storing pointer in last character string)
strtod	Conversion of character string to floating-point number (double type) (storing pointer in last character string
rand	Pseudorandom number sequence generation
srand	Setting of type of pseudorandom number sequence
abort	Terminates the program

abs

Output absolute value (int type)

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
int abs(int *j*);

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |.

[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*.

[Example]

```
#include    <stdlib.h>
void func(int l) {
        int val;
        val = -15;
        l = abs(val);   /*Returns absolute value of val, 15, to 1.*/
}
```

labs

Output absolute value (long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long    labs(long j);
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |.

[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.

llabs

Output absolute value (long long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long       llabs(long long j);
```

[Return value]

Returns the absolute value of *j* (size of *j*), | *j* |.

[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 long type instead of int type, and the return value is also of long long type.

[Caution]

This function cannot be used when the -lang=c option and -strict_std option are both specified.

bsearch

Binary search

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void*    bsearch(const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *,
                                                                                    const void*));
```

[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 array must be arranged in the ascending order in respect to the compare function indicated by *compar* (last argument). Define the compare function indicated by *compar* to have two arguments. If the first argument is less than the second, a negative integer must be returned as the result. If the two arguments coincide, zero must be returned. If the first is greater than the second, a positive integer must be returned.

[Example]

```
#include    <stdlib.h>
#include    <string.h>
int compar(const void *x, const void *y);

void func(void) {
        static  char    *base[] = {"a", "b", "c", "d", "e", "f"};
        char            *key = "c";     /*Search key is "c".*/
        char            **ret;
                                        /*Pointer to "c" is stored in ret.*/
        ret = (char **) bsearch((char *) &key, (char *) base, 6, sizeof(char *), com-
par);
}
int compar(const void *x, const void *y) {
       return(strcmp(x, y));          /*Returns positive, zero, or negative integer as
                                        result of comparing arguments.*/
}
```

qsort

Sort

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    qsort(void *base, size_t nmemb, size_t size, int (*compar)(const void*, const void *));
```

[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.

div

Division (int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
div_t   div(int n, int d);
```

[Return value]

The structure storing the result of the division is returned.

[Description]

This function is used to divide a value of int type

This function calculates the quotient and remainder resulting from dividing numerator *n* by denominator *d*, and stores these two integers as the members of the following structure div_t.

```
typedef struct {
        int quot;
        int rem;
} div_t;
```

quot the quotient, and rem is the remainder. If *d* is not zero, and if "*r* = div(*n*, *d*);", *n* is a value equal to "*r*.rem + *d* * *r*.quot".

If *d* is zero, the resultant quot member has a sign the same as *n* and has the maximum size that can be expressed. The rem member is 0.

[Example]

```
#include    <stdlib.h>
void func(void) {
        div_t   r;
        r = div(110, 3);    /*36 is stored in r.quot, and 2 is stored in r.rem.*/
}
```

ldiv

Division (long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
ldiv_t  ldiv(long n, long d);
```

[Return value]

The structure storing the result of the division is returned.

[Description]

This function is used to divide a value of long type.

This function calculates the quotient and remainder resulting from dividing numerator *n* by denominator *d*, and stores these two integers as the members of the following structure div_t.

```
typedef struct {
        long    quot;
        long    rem;
} ldiv_t;
```

quot the quotient, and rem is the remainder. If *d* is not zero, and if "*r* = div(*n*, *d*);", *n* is a value equal to "*r*.rem + *d* * *r*.quot".

If *d* is zero, the resultant quot member has a sign the same as *n* and has the maximum size that can be expressed. The rem member is 0.

lldiv

Division (long long type)

[Classification]

Standard library

[Syntax]

#include <stdlib.h>
lldiv_t lldiv(long long *n*, long long *d*);

[Return value]

The structure storing the result of the division is returned.

[Description]

This function is used to divide a value of long long type.
This function calculates the quotient and remainder resulting from dividing numerator *n* by denominator *d*, and stores these two integers as the members of the following structure div_t.

```
typedef struct {
        long long   quot;
        long long   rem;
} lldiv_t;
```

quot the quotient, and rem is the remainder. If *d* is not zero, and if "*r* = div(*n*, *d*);", *n* is a value equal to "*r*.rem + *d* * *r*.quot".
If *d* is zero, the resultant quot member has a sign the same as *n* and has the maximum size that can be expressed. The rem member is 0.

[Caution]

This function cannot be used when the -lang=c option and -strict_std option are both specified.

atoi

Conversion of character string to integer (int type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
int atoi(const char *str);
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.

[Description]

This function converts the first part of the character string indicated by *str* into an int type representation. atoi is the same as "(int) strtol (*str*, NULL, 10)".

atol

Conversion of character string to integer (long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long    atol(const char *str);
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.

[Description]

This function converts the first part of the character string indicated by *str* into a long int type representation. atol is the same as "strtol (*str*, NULL, 10)".

atoll

Conversion of character string to integer (long long type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long   atoll(const char *str);
```

[Return value]

Returns the converted value if the partial character string could be converted. If it could not, 0 is returned.

[Description]

This function converts the first part of the character string indicated by *str* into a long long int type representation. atol is the same as "strtol (*str*, NULL, 10)".

[Caution]

This function cannot be used when the -lang=c option and -strict_std option are both specified.

strtol

Conversion of character string to integer (long type) and storing pointer in last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long    strtol(const char *str, char **ptr, int base);
```

[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 (because the converted value is too great), LONG_MAX or LONG_MIN is returned, and macro ERANGE is set to global variable errno.

[Description]

This function converts the first part of the character string indicated by *str* into a long type representation. strol first divides the input characters into the following three parts: the "first blank", "a string represented by the *base* number determined by the value of base and is subject to conversion into an integer", and "the last one or more character string that is not recognized (including the null character (\0))". Then strtol converts the string into an integer, and returns the result.

(1) Specify 0 or 2 to 36 as argument *base*.

(a) If *base* is 0
The expected format of the character string subject to conversion is of integer format having an optional + or - sign and "0x", indicating a hexadecimal number, prefixed.

(b) If the value of *base* is 2 to 36
The expected format of the character string is of character string or numeric string type having an optional + or - sign prefixed and expressing an integer whose base is specified by base. Characters "a" (or "A") through "z" (or "Z") are assumed to have a value of 10 to 35. Only characters whose value is less than that of base can be used.

(c) If the value of *base* is 16
"0x" is prefixed (suffixed to the sign if a sign exists) to the string of characters and numerals (this can be omitted).

(2) The string subject to conversion is defined as the longest partial string at the beginning of the input character string that starts with the first character other than blank and has an expected format.

(a) If the input character string is vacant, if it consists of blank only, or if the first character that is not blank is not a sign or a character or numeral that is permitted, the subject string is vacant.

(b) If the string subject to conversion has an expected format and if the value of *base* is 0, the base number is judged from the input character string. The character string led by 0x is regarded as a hexadecimal value, and the character string to which 0 is prefixed but x is not is regarded as an octal number. All the other character strings are regarded as decimal numbers.

(c) If the value of *base* is 2 to 36, it is used as the base number for conversion as mentioned above.

(d) If the string subject to conversion starts with a - sign, the sign of the value resulting from conversion is reversed.

(3) The pointer that indicates the first character string

(a) This is stored in the object indicated by *ptr*, if *ptr* is not a null pointer.

(b) If the string subject conversion is vacant, or if it does not have an expected format, conversion is not executed. The value of *str* is stored in the object indicated by *ptr* if ptr is not a null pointer.

Remark This function is not reentrancy.

[Example]

```
#include    <stdlib.h>
void func(long ret) {
        char    *p;
        ret = strtol("10", &p, 0);      /*10 is returned to ret.*/
        ret = strtol("0x10", &p, 0);    /*16 is returned to ret.*/
        ret = strtol("10x", &p, 2);     /*2 is returned to ret, and pointer to "x" is
                                          returned to area of p.*/
        ret = strtol("2ax3", &p, 16);   /*42 is returned to ret, and pointer to "x" is
                                          returned to area of p.*/
          :
}
```

strtoul

Conversion of character string to integer (unsigned long type) and storing pointer in last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
unsigned long   strtoul(const char *str, char **ptr, int base);
```

[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, ULONG_MAX is returned, and macro ERANGE is set to global variable errno.

[Description]

This function is the same as strtol except that the type of the return value is of unsigned long type.

strtoll

Conversion of character string to integer (long long type) and storing pointer in last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
long long   strtoll(const char *str, char **ptr, int base);
```

[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 (the converted value is too larger), LLONG_MAX or LLONG_MIN is returned, and macro ERANGE is set to global variable errno.

[Description]

This function is the same as strtol except that the type of the return value is of long long type.

[Caution]

This function cannot be used when the -lang=c option and -strict_std option are both specified.

strtoull

Conversion of character string to integer (unsigned long long type) and storing pointer in last character string

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
unsigned long long  strtoull(const char *str, char **ptr, int base);
```

[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, ULLONG_MAX is returned, and macro ERANGE is set to global variable errno.

[Description]

This function is the same as strtol except that the type of the return value is of unsigned long long type.

[Caution]

This function cannot be used when the -lang=c option and -strict_std option are both specified.

atoff

Conversion of character string to floating-point number (float type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
float   atoff(const char *str);
```

[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 (the value is not in the range in which it can be expressed), HUGE_VAL or -HUGE_VAL is returned, and ERANGE is set to global variable errno. If an underflow occurs, 0 is returned, and macro ERANGE is set to global variable errno.

[Description]

This function converts the first portion of the character string indicated by *str* into a float type representation. atoff is the same as "strtodf (*str*, NULL)".

atof

Conversion of character string to floating-point number (double type)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
double  atof(const char *str);
```

[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 (the value is not in the range in which it can be expressed), HUGE_VAL or -HUGE_VAL is returned, and ERANGE is set to global variable errno. If an underflow occurs, 0 is returned, and macro ERANGE is set to global variable errno.

[Description]

This function converts the first portion of the character string indicated by *str* into a float type representation. atoff is the same as "strtod (*str*, NULL)".

strtodf

Conversion of character string to floating-point number (float type) (storing pointer in last character string)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
float   strtodf(const char *str, char **ptr);
```

[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 (the value is not in the range in which it can be expressed), HUGE_VAL or -HUGE_VAL is returned, and ERANGE is set to global variable errno. If an underflow occurs, 0 is returned, and macro ERANGE is set to global variable errno.

[Description]

This function converts the first part of the character string indicated by *str* into a float type representation. The part of the character string to be converted is in the following format and is at the beginning of *str* with the maximum length, starting with a normal character that is not a space.

[+ | -] digits [.] [digits] [(e | E) [+ | -] digits]

If *str* is vacant or consists of space characters only, if the first normal character is other than "+", "-", ".", or a numeral, the partial character string does not include a character. If the partial character string is vacant, conversion is not executed, and the value of str is stored in the area indicated by *ptr*. If the partial character string is not vacant, it is converted, and a pointer to the last character string (including the null character (\0) indicating at least the end of str) is stored in the area indicated by *ptr*.

Remark This function is not reentrancy.

[Example]

```
#include    <stdlib.h>
#include    <stdio.h>
void func(float ret) {
        char    *p, *str, s[30];
        str = "+5.32a4e";
        ret = strtodf(str, &p);         /*5.320000 is returned to ret, and pointer to
                                          "a" is stored in area of p.*/
        sprintf(s, "%lf\t%c", ret, *p); /*"5.320000 a" is stored in array indicated
                                          by s.*/
}
```

strtod

Conversion of character string to floating-point number (double type) (storing pointer in last character string)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
double  strtod(const char *str, char **ptr);
```

[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 (the value is not in the range in which it can be expressed), HUGE_VAL or -HUGE_VAL is returned, and ERANGE is set to global variable errno. If an underflow occurs, 0 is returned, and macro ERANGE is set to global variable errno.

[Description]

This function converts the first part of the character string indicated by *str* into a float type representation. The part of the character string to be converted is in the following format and is at the beginning of *str* with the maximum length, starting with a normal character that is not a space.

[+ | -] digits [.] [digits] [(e | E) [+ | -] digits]

If *str* is vacant or consists of space characters only, if the first normal character is other than "+", "-", ".", or a numeral, the partial character string does not include a character. If the partial character string is vacant, conversion is not executed, and the value of str is stored in the area indicated by *ptr*. If the partial character string is not vacant, it is converted, and a pointer to the last character string (including the null character (\0) indicating at least the end of str) is stored in the area indicated by *ptr*.

Remark This function is not reentrancy.

```
#include    <stdlib.h>
typedef struct {
        double  d[3];
        int     i[2];
} s_data;
int func(void) {
        sdata   *buf;
        if((buf = calloc(40, sizeof(s_data))) == NULL)  /*allocate an area for 40
                                                          s_data*/
                return(1);
           :
        free(buf);                                      /*release the area*/
        return(0);
}
```

rand

Pseudorandom number sequence generation

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
int 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.

[Example]

```
#include    <stdlib.h>
void func(void) {
        if((rand() & 0xF) < 4)
                func1();    /*execute func1 with a probability of 25%*/
}
```

srand

Setting of type of pseudorandom number sequence

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    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.

abort

Terminates the program

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void   abort(void);
```

[Description]

Calling abort(void) terminates the program. An abort function that suits the user system must be created in advance.

[Example]

```
#include    <assert.h>
int func(void);
int main() {
    int ret;
    ret = func();
    if (ret == 0) {
    abort();       <- abort() is called if ret is not 0
    }
    return 0;
}
```

7.4.9 Non-local jump functions

Non-local jump functions are as follows.

Table 7.12 Non-Local Jump Functions

Function/Macro Name	Outline
longjmp	Non-local jump
setjmp	Set destination of non-local jump

longjmp

Non-local jump

[Classification]

Standard library

[Syntax]

#include <setjmp.h>
void 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. *val* as a return value for setjmp.

[Definition of jmp_buf type in setjmp.h]

```
typedef int     jmp_buf[14];
```

[Caution]

When this function is called, only data in the registers reserved by the compiler are saved and restored.
If setjmp is called from within a function in the 22-register mode or common-register mode, data in r20 to r24 are destroyed from within a function in the 32-register mode, and longjmp is then called, the values of r20 to r24 will not be recoverable. In such cases, the values of r20 to r24 must be restored before longjmp is called if they are required.
When -Xep=fix is specified, ep/fix/libsetjmp.lib must be used.

[Example]

```
#include    <setjmp.h>
#define ERR_XXX1    1
#define ERR_XXX2    2

jmp_buf jmp_env;

void main(void) {
    for(;;) {
        switch(setjmp(jmp_env)) {
            case    ERR_XXX1:
                /*termination of error XXX1*/
                break;
            case    ERR_XXX2:
                /*termination of error XXX2*/
                break;
            case    0:          /*no non-local jumps*/
            default:
                break;
        }
    }
}

void funcXXX(void) {
    longjmp(jmp_env, ERR_XXX1); /*Non-local jumps are performed upon generation of
                                  error XXX1.*/
    longjmp(jmp_env, ERR_XXX2); /*Non-local jumps are performed upon generation of
                                  error XXX2.*/
}
```

setjmp

Set destination of non-local jump

[Classification]

Standard library

[Syntax]

```
#include <setjmp.h>
int 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*.

[Definition of jmp_buf type in setjmp.h]

```
typedef int     jmp_buf[14];
```

[Caution]

When this function is called, only data in the registers reserved by the compiler are saved and restored.
If setjmp is called from within a function in the 22-register mode or common-register mode, data in r20 to r24 are destroyed from within a function in the 32-register mode, and longjmp is then called, the values of r20 to r24 will not be recoverable. In such cases, the values of r20 to r24 must be restored before longjmp is called if they are required.
When -Xep=fix is specified, ep/fix/libsetjmp.lib must be used.
Do not call the setjmp function indirectly using a pointer.

7.4.10 Mathematical functions

Mathematical functions are as follows.

Table 7.13 Mathematical Functions

Function/Macro Name	Outline
acos functions	Arc cosine
asin functions	Arc sine
atan functions	Arc tangent
atan2 functions	Arc tangent (y / x)
cos functions	Cosine
sin functions	Sine
tan functions	Tangent
cosh functions	Hyperbolic cosine
sinh functions	Hyperbolic sine
tanh functions	Hyperbolic tangent
exp functions	Exponent function
frexp functions	Divide floating-point number into mantissa and power
ldexp functions	Convert floating-point number to power
log functions	Logarithmic function (natural logarithm)
log10 functions	Logarithmic function (base = 10)
modf functions	Divide floating-point number into integer and decimal
fabs functions	Absolute value function
pow functions	Power function
sqrt functions	Square root function
ceil functions	ceiling function
floor functions	floor function
round functions [V2.01.00 or later]	Rounds a floating-point number to the nearest integer in the floating-point representation
lround functions and llround functions [V2.01.00 or later]	Rounds a floating-point number to the nearest integer
trunc functions [V2.01.00 or later]	Rounds a floating-point number to the nearest integer in the floating-point representation
fmod functions	Remainder function
copysign functions [V2.00.00 or later]	Combine sign and absolute value
fmax functions [V2.00.00 or later]	Choose a greater value
fmin functions [V2.00.00 or later]	Choose a less value

acos functions

[Syntax]

double acos(double x)
float acosf(float x)
long double acosl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the arc cosine of x.

[Special case]

Condition	Return value	Exception
|x|>1	NaN	EDOM

asin functions

[Syntax]

double asin(double x)
float asinf(float x)
long double asinl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the arc sine of x.

[Special case]

Condition	Return value	Exception
|x|>1	NaN	EDOM

atan functions

[Syntax]

double atan(double x)
float atanf(float x)
long double atanl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the arc tangent of x.

[Special case]

Condition	Return value	Exception
Underflow occurred	-	ERANGE

atan2 functions

[Syntax]

double atan2(double y, double x)
float atan2f(float y, float x)
long double atan2l(long double y, long double x) [V2.01.00 or later]

[Description]

These functions calculate the arc tangent of y/x.

[Special case]

Condition	Return value	Exception
x==0, y==0	NaN	EDOM
x==±∞, y==±∞	NaN	EDOM
Underflow occurred	±0	ERANGE
x<0, y==0	π	-
x==0	±π/2	-

cos functions

[Syntax]

double cos(double x)
float conf(float x)
long double cosl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the cosine of x (measured in radians).

[Special case]

Condition	Return value	Exception
x==±∞	NaN	EDOM

sin functions

[Syntax]

double sin(double x)
float sinf(float x)
long double sinl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the sine of x (measured in radians).

[Special case]

Condition	Return value	Exception
x==±∞	NaN	EDOM
Underflow occurred	-	ERANGE

tan functions

[Syntax]

double tan(double x)
float tanf(float x)
long double tanl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the tangent of x (measured in radians).

[Special case]

Condition	Return value	Exception
x==±∞	NaN	EDOM
Underflow occurred	-	ERANGE

cosh functions

[Syntax]

double cosh(double x)
float coshf(float x)
long double coshl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the hyperbolic cosine of x.

[Special case]

Condition	Return value	Exception
Overflow occurred	HUGE_VAL	ERANGE

sinh functions

[Syntax]

double sinh(double x)
float sinhf(float x)
long double sinhl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the hyperbolic sine of x.

[Special case]

Condition	Return value	Exception
Overflow occurred	HUGE_VAL	ERANGE

tanh functions

[Syntax]

double tanh(double x)
float tanhf(float x)
long double tanhl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the hyperbolic tangent of x.

[Special case]

Condition	Return value	Exception
Underflow occurred	-	ERANGE

exp functions

[Syntax]

double exp(double x)
float expf(float x)
long double expl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the base-e (base of natural logarithm) exponential of x.

[Special case]

Condition	Return value	Exception
Underflow occurred	-	ERANGE
Overflow occurred	HUGE_VAL	ERANGE

frexp functions

[Syntax]

double frexp(double x, int *exp)
float frexpf(float x, int *exp)
long double frexpl(long double x, int *exp) [V2.01.00 or later]

[Description]

These functions divide x into a normalized number and an integral power of 2. They return the normalized number and store the integer in the int object pointed to by exp.

Assuming that the returned value is ret, it satisfies the following conditions.

- 0.5 <= |ret| < 1
- $x = ret * 2^{exp}$

[Special case]

Condition	Return value	Exception
x==0	0, *exp=0	-
x==±∞	NaN, *exp=0	EDOM
x==NaN	NaN, *exp=0	-

ldexp functions

[Syntax]

double ldexp(double x)
float ldexpf(float x)
long double ldexpl(long double x) [V2.01.00 or later]

[Description]

These functions multiply a floating-point number by an integral power of 2.

[Special case]

Condition	Return value	Exception
Overflow occurred	±HUGE_VAL	ERANGE
Underflow occurred	Denormal number	ERANGE

log functions

[Syntax]

double log(double x)
float logf(float x)
long double logl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the natural logarithm of x with e as the base.

[Special case]

Condition	Return value	Exception
x<0	NaN	EDOM
x==0	-∞	ERANGE

log10 functions

[Syntax]

double log10(double x)
float log10f(float x)
long double log10l(long double x) [V2.01.00 or later]

[Description]

These functions calculate the common logarithm of x with 10 as the base.

[Special case]

Condition	Return value	Exception
x<0	NaN	EDOM
x==0	-∞	ERANGE

modf functions

[Syntax]

double modf(double x, double *iptr)
float modff(float x, float *iptr)
long double modfl(long double x, long double *iptr) [V2.01.00 or later]

[Description]

These functions divide x into integral and fractional parts. They return the fractional part and store the integral part in the object pointed to by iptr.

Both the integral part and fractional part have the same sign as x.

fabs functions

[Syntax]

double fabs(double x)
float fabsf(float x)
long double fabsl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the absolute value of x.

pow functions

[Syntax]

double pow(double x)
float powf(float x)
long double powl(long double x) [V2.01.00 or later]

[Description]

These functions calculate x raised to the power y.

[Special case]

Condition	Return value	Exception
x<0 and y is a non-integer.	NaN	EDOM
x<0, y==∞	NaN	EDOM
x==±∞, y==0	NaN	EDOM
x==0, y==0	NaN	EDOM
x==0, y<0	+HUGE_VAL	ERANGE
Overflow occurred	±HUGE_VAL	ERANGE
Underflow occurred	0	ERANGE

sqrt functions

[Syntax]

double sqrt(double x)
float sqrtf(float x)
long double sqrtl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the square root of x.

[Special case]

Condition	Return value	Exception
x<0	NaN	EDOM

ceil functions

[Syntax]

double ceil(double x)
float ceilf(float x)
long double ceill(long double x) [V2.01.00 or later]

[Description]

These functions calculate the smallest integer value that is not less than x.

floor functions

[Syntax]

double floor(double x)
float floorf(float x)
long double floorl(long double x) [V2.01.00 or later]

[Description]

These functions calculate the largest integer value that is not greater than x.

round functions [V2.01.00 or later]

[Syntax]

```
double round(double x)
float roundf(float x)
long double roundl(long double x)
```

[Description]

These functions round x to the nearest integer value. If x is just in the middle, the value farther away from 0 is selected, regardless of the current rounding direction.

lround functions and llround functions [V2.01.00 or later]

[Syntax]

long int lround(double x)
long int lroundf(float x)
long int lroundl(long double x)
long long int llround(double x)
long long int llroundf(float x)
long long int llroundl(long double x)

[Description]

These functions round x to the nearest integer value. If x is just in the middle, the value in the direction farther away from 0 is selected, regardless of the current rounding direction.

[Special case]

Condition	Return value	Exception
x==NaN	0	EDOM
x==±∞	0	EDOM

trunc functions [V2.01.00 or later]

[Syntax]

```
double trunc(double x)
float truncf(float x)
long double truncl(long double x)
```

[Description]

These functions round x to the nearest integer value (however, its absolute value must not be larger than the absolute value of x).

fmod functions

[Syntax]

double fmod(double x, double y)
float fmodf(float x, float y)
long double fmodl(long double x, long double y) [V2.01.00 or later]

[Description]

These functions return the value of "x - ny" for integer n, when y is a value other than 0. The result has the same sign as x, and its absolute value must be less than the absolute value of y.

[Special case]

Condition	Return value	Exception
x==±∞	NaN	EDOM
y==±∞	x	-
y==0	NaN	EDOM

copysign functions [V2.00.00 or later]

[Syntax]

double copysign(double x, double y)
float copysignf(float x, float y)
long double copysignl(long double x, long double y) [V2.01.00 or later]

[Description]

These functions return a value with the absolute value of x and the sign of y.

fmax functions [V2.00.00 or later]

[Syntax]

double fmax(double x, double y)
float fmaxf(float x, float y)
long double fmaxl(long double x, long double y) [V2.01.00 or later]

[Description]

These functions return the larger value among x and y. If either x or y is a NaN, the other value is returned.

fmin functions [V2.00.00 or later]

[Syntax]

```
double fmin(double x, double y)
float fminf(float x, float y)
long double fminl(long double x, long double y) [V2.01.00 or later]
```

[Description]

These functions return the smaller value among x and y. If either x or y is a NaN, the other value is returned.

7.4.11 RAM section initialization function

RAM section initialization function are as follows.

Table 7.14 RAM Section Initialization Function

Function/Macro Name	Outline
_INITSCT_RH	Copies initial values to or clears sections in RAM

_INITSCT_RH

Copies initial values to or clears sections in RAM

[Classification]

Standard library

[Syntax]

```
#include <_h_c_lib.h>
void _INITSCT_RH(void * datatbl_start, void * datatbl_end, void * bsstbl_start, void * bsstbl_end)
```

[Argument(s)/Return value]

Argument	Return Value
datatbl_start : First address of the initialization table for a section with the data attribute datatbl_end: Last address of the initialization table for a section with the data attribute bsstbl_start : First address of the initialization table for a section with the bss attribute bsstbl_end: Last address of the initialization table for a section with the bss attribute	None

[Description]

For sections in RAM, this function copies initial values for a section with the data attribute from the ROM area and clears a section with the bss attribute to 0.

The first and second parameters are used to pass the first and last addresses of the initialization table for a section with the data attribute.

The third and fourth parameters are used to pass the first and last addresses of the initialization table for a section with the bss attribute.

If the value of the first parameter is greater than or equal to that of the second parameter, the section with the data attribute is not initialized.

If the value of the third parameter is greater than or equal to that of the fourth parameter, the section with the bss attribute is not cleared to zero.

[Example]

```
struct {
    void *rom_s; //The first address of the section with the data attribute in the ROM
    void *rom_e; //The last address of the section with the data attribute in the ROM
    void *ram_s; //The first address of the section with the data attribute in the RAM
} _C_DSEC[M];

struct {
    void *bss_s; //The first address of the section with the bss attribute in the RAM
    void *bss_e; //The last address of the section with the bss attribute in the RAM
} _C_BSEC[N];

_INITSCT_RH(_C_DSEC, _C_DSEC + M, _C_BSEC, _C_BSEC + N);
```

Remark When the start address of the .bss section is 0x100 and the size of the section is 0x50 bytes, the memory addresses that are actually cleared to 0 are 0x100, 0x101, ..., 0x14e, and 0x14f but specify addresses 0x100 and 0x150 in the initialization table.

7.4.12 Peripheral device initialization function

The peripheral device initialization function is as follows.

Table 7.15 Peripheral Device Initialization Function

Function/Macro Name	Outline
hdwinit	Initialization of peripheral devices immediately after the CPU reset

hdwinit

Initialization of peripheral devices immediately after the CPU reset.

[Classification]

Standard library

[Syntax]

```
void    hdwinit(void);
```

[Description]

The peripheral device initialization 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.

7.4.13 Operation runtime functions

Operation runtime functions are as follows.

Table 7.16 Operation Runtime Functions

Classification	Function Name	Outline
float type operation function	_COM_fadd	Addition of single-precision floating-point
	_COM_fsub	Subtraction of single-precision floating-point
	_COM_fmul	Multiplication of single-precision floating-point
	_COM_fdiv	Division of single-precision floating-point
double type operation function	_COM_dadd	Addition of double-precision floating-point
	_COM_dsub	Subtraction of double-precision floating-point
	_COM_dmul	Multiplication of double-precision floating-point
	_COM_ddiv	Division of double-precision floating-point
long long type operation function	_COM_mul64	Multiplication of 64-bit integer
	_COM_div64	Division of signed 64-bit integer
	_COM_udiv64	Division of unsigned 64-bit integer
	_COM_rem64	Remainder of signed 64-bit integer
	_COM_urem64	Remainder of unsigned 64-bit integer
	_COM_shll_64_32	Logical left shift of 64-bit integer
	_COM_shrl_64_32	Logical right shift of 64-bit integer
	_COM_shra_64_32	Arithmetic right shift 64-bit integer
	_COM_neg64	Sign inversion

Classification	Function Name	Outline
Type conversion function	_COM_itof	Conversion from 32-bit integer to single-precision floating-point number
	_COM_itod	Conversion from 32-bit integer to double-precision floating-point number
	_COM_utof	Conversion from unsigned 32-bit integer to single-precision floating-point number
	_COM_utod	Conversion from unsigned 32-bit integer to double-precision floating-point number
	_COM_i64tof	Conversion from 64-bit integer to single-precision floating-point number
	_COM_i64tod	Conversion from 64-bit integer to double-precision floating-point number
	_COM_u64tof	Conversion from unsigned 64-bit integer to single-precision floating-point number
	_COM_u64tod	Conversion from unsigned 64-bit integer to double-precision floating-point number
	_COM_ftoi	Conversion from single-precision floating-point number to 32-bit integer
	_COM_dtoi	Conversion from double-precision floating-point number to 32-bit integer
	_COM_ftou	Conversion from single-precision floating-point number to unsigned 32-bit integer
	_COM_dtou	Conversion from double-precision floating-point number to unsigned 32-bit integer
	_COM_ftoi64	Conversion from single-precision floating-point number to 64-bit integer
	_COM_dtoi64	Conversion from double-precision floating-point number to 64-bit integer
	_COM_ftou64	Conversion from single-precision floating-point number to unsigned 64-bit integer
	_COM_dtou64	Conversion from double-precision floating-point number to unsigned 64-bit integer
	_COM_ftod	Conversion from single-precision floating-point number to double-precision floating-point number
	_COM_dtof	Conversion from double-precision floating-point number to single-precision floating-point number

Classification	Function Name	Outline
Floating-point comparison functions	_COM_fgt	Comparison
	_COM_fge	Comparison
	_COM_feq	Comparison
	_COM_fne	Comparison
	_COM_flt	Comparison
	_COM_fle	Comparison
	_COM_funord	Incomparable
	_COM_dgt	Comparison
	_COM_dge	Comparison
	_COM_deq	Comparison
	_COM_dne	Comparison
	_COM_dlt	Comparison
	_COM_dle	Comparison
	_COM_dunord	Incomparable

7.4.14 Checks for indirect function calls function

Checks for indirect function calls function is as follows.

Table 7.17 Checks for indirect function calls function

Function/Macro Name	Outline
__control_flow_integrity	Checks for indirect function calls.

7.4.15 Dynamic memory management functions

Dynamic memory management functions are as follows.

Table 7.18 Dynamic memory management functions

Function/Macro Name	Outline
calloc	Dynamic memory allocation
free	Dynamic memory release
malloc	Dynamic memory allocation
realloc	Dynamic memory re-allocation

calloc

Memory allocation (initialized to zero)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    *calloc(size_t nmemb, size_t size);
```

[Return value]

When area allocation succeeds, a pointer to that area is returned. When the area could not be allocated, a null pointer is returned.

[Description]

This function allocates an area for an array of *nmemb* elements. The allocated area is initialized to zeros.

[Caution]

The memory area management functions automatically allocate memory area as necessary from the heap memory area.

Also, the size of the default is 0x1000 bytes, so when it's changed, the heap memory area must be allocated. The area allocation should be performed first by an application.

[Heap memory setup example]

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x1000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark 1. The variable "_REL_sysheap" points to the starting address of heap memory. This value must be a multiple of 4.

Remark 2. The required heap memory size (bytes) should be set for the variable "_REL_sizeof_sysheap".

[Example]

```
#include    <stdlib.h>
typedef struct {
    double  d[3];
    int     i[2];
} s_data;
int func(void) {
    sdata   *buf;
    if((buf = calloc(40, sizeof(s_data))) == NULL)  /*allocate an area for 40 s_data*/
        return(1);
      :
    free(buf);                                      /*release the area*/
    return(0);
}
```

free

Memory release

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    free(void *ptr);
```

[Description]

This function releases the area pointed to by *ptr* so that this area is subsequently available for allocation. The area that was acquired by calloc, malloc, or realloc must be specified for *ptr*.

[Example]

```
#include    <stdlib.h>
typedef struct {
    double  d[3];
    int     i[2];
} s_data;
int func(void) {
    s_data   *buf;
    if((buf = calloc(40, sizeof(s_data))) == NULL)  /*allocate an area for 40 s_data*/
        return(1);
      :
    free(buf);                                      /*release the area*/
    return(0);
}
```

If one of the following operations is performed when using the library for the security facility, the __heap_chk_fail function is called.

- 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.
- A value is written to up to four bytes before and after the area allocated by calloc, malloc, or realloc 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 only possible type of return value is void and the __heap_chk_fail function does not have formal parameters.
- Do not define the function as static.
- Corruption of heap memory area should not be detected recursively in the __heap_chk_fail function.
- PIC (see "8.6 PIC/PID Facility") must not be performed for 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.

```
#include <stdlib.h>

void sub(int *ip) {
    ...
    free(ip);
}

int func(void) {
    int *ip;
    if ((ip = malloc(40 * sizeof(int))) == NULL)
        if ((ip = malloc(10 * sizeof(int))) == NULL) return(1);
        else sub(ip); /* First appearance of free */
    else
        ...
    free(ip); /* Second appearance of free */
    return(0);
}

void __heap_chk_fail(void) {
    /* Processing when corruption of heap memory area is detected */
}
```

malloc

Memory allocation(not initialized to zero)

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    *malloc(size_t size);
```

[Return value]

When area allocation succeeds, a pointer to that area is returned. When the area could not be allocated, a null pointer is returned.

[Description]

This function allocates an area having a size indicated by *size*. The area is not initialized.

[Caution]

The memory area management functions automatically allocate memory area as necessary from the heap memory area.

Also, the size of the default is 0x1000 bytes, so when it's changed, the heap memory area must be allocated. The area allocation should be performed first by an application.

[Heap memory setup example]

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x1000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark 1. The variable "_REL_sysheap" points to the starting address of heap memory. This value must be a multiple of 4.

Remark 2. The required heap memory size (bytes) should be set for the variable "_REL_sizeof_sysheap".

realloc

Memory re-allocation

[Classification]

Standard library

[Syntax]

```
#include <stdlib.h>
void    *realloc(void *ptr, size_t size);
```

[Return value]

When area allocation succeeds, a pointer to that area is returned. When the area could not be allocated, a null pointer is returned.

[Description]

This function changes the size of the area pointed to by *ptr* to the size indicated by *size*. The contents of the area are unchanged up to the smaller of the previous size and the specified *size*. If the area is expanded, the contents of the area greater than the previous size are not initialized. When *ptr* is a null pointer, the operation is the same as that of malloc (*size*). Otherwise, the area that was acquired by calloc, malloc, or realloc must be specified for *ptr*.

[Caution]

The memory area management functions automatically allocate memory area as necessary from the heap memory area.

Also, the size of the default is 0x1000 bytes, so when it's changed, the heap memory area must be allocated. The area allocation should be performed first by an application.

[Heap memory setup example]

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x1000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

Remark 1. The variable "_REL_sysheap" points to the starting address of heap memory. This value must be a multiple of 4.

Remark 2. The required heap memory size (bytes) should be set for the variable "_REL_sizeof_sysheap".

7.5 Usage of Data Sections and List of Reentrancy

This chapter explains the usage of constant data sections (.const), data sections with initial value (.data), and data sections without initial value (.bss) by various functions which are included in libraries, and details of reentrancy.

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reentrancy	Remarks (library for storage, cause of non-reentrancy)
strpbrk	X	X	X	O	
strrchr	X	X	X	O	
strchr	X	X	X	O	
strstr	X	X	X	O	
strspn	X	X	X	O	
strcspn	X	X	X	O	
strcmp	X	X	X	O	
strncmp	X	X	X	O	
strcpy	X	X	X	O	
strncpy	X	X	X	O	
strcat	X	X	X	O	
strncat	X	X	X	O	
strtok	X	X	O	X	Internal management data
strlen	X	X	X	O	
strerror	O	O	X	X	Internal management data
memchr	X	X	X	O	
memcmp	X	X	X	O	
memcpy	X	X	X	O	
memmove	X	X	X	O	
memset	X	X	X	O	
toupper	O	X	X	O	
tolower	O	X	X	O	
isalnum	O	X	X	O	
isalpha	O	X	X	O	
isascii	X	X	X	O	
isupper	O	X	X	O	
islower	O	X	X	O	
isdigit	O	X	X	O	
isxdigit	O	X	X	O	
iscntrl	O	X	X	O	
ispunct	O	X	X	O	
isspace	O	X	X	O	

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
isprint	O	X	X	O	
isgraph	O	X	X	O	
fread	X	X	X	O	
getc	X	X	X	O	
fgetc	X	X	X	O	
fgets	X	X	X	O	
fwrite	X	X	X	O	
putc	X	X	X	O	
fputc	X	X	X	O	
fputs	X	X	X	O	
getchar	X	O	X	X	stdin
gets	X	O	X	X	stdin
putchar	X	O	X	X	stdout
puts	X	O	X	X	stdout
sprintf	O	X	O	X	errno
fprintf	O	X	O	X	errno
vsprintf	O	X	O	X	errno
printf	O	O	O	X	errno, stdout
vfprintf	O	X	O	X	errno
vprintf	O	O	O	X	errno, stdout
sscanf	O	X	X	O	
fscanf	O	X	X	O	
scanf	O	O	X	X	stdin
ungetc	X	X	X	O	
rewind	X	X	X	O	
perror	O	O	O	X	errno,stderr
abs	X	X	X	O	
labs	X	X	X	O	
llabs	X	X	X	O	
bsearch	X	X	X	O	
qsort	X	X	X	O	
div	X	X	X	O	
ldiv	X	X	X	O	
lldiv	X	X	X	O	

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
lldiv	O	X	X	O	(libc.lib)
atoi	O	X	O	X	errno
atol	O	X	O	X	errno
atoll	O	X	O	X	errno
strtol	O	X	O	X	errno
strtoul	O	X	O	X	errno
strtoll	O	X	O	X	errno
strtoull	O	X	O	X	errno
atoff	O	X	O	X	errno
atof	O	X	O	X	errno
strtodf	O	X	O	X	errno
strtod	O	X	O	X	errno
rand	X	O	X	X	Internal management data
srand	X	O	X	X	Internal management data
abort	X	X	X	-	Processing is not returned
longjmp	X	X	X	X	SP
setjmp	X	X	X	O	
expf	O	X	O	X	errno
exp	O	X	O	X	errno
expl	O	X	O	X	errno
logf	O	X	O	X	errno
log	X	X	O	X	errno
log	O	X	O	X	(libm.lib, softfloat\libm.lib) errno
logl	O	X	O	X	errno
logl	X	X	O	X	(rhf8n.lib, rhf8z.lib, libm.lib) errno
log10f	X	X	O	X	errno
log10f	O	X	O	X	(libmf.lib, softfloat\libmf.lib) errno
log10	X	X	O	X	errno
log10	O	X	O	X	(libm.lib, softfloat\libm.lib) errno
log10l	O	X	O	X	errno
log10l	X	X	O	X	(rhf8n.lib, rhf8z.lib, libm.lib) errno

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
powf	O	X	O	X	errno
pow	O	X	O	X	errno
powl	O	X	O	X	errno
sqrtf	X	X	O	X	errno
sqrtf	O	X	O	X	(libmf.lib, softfloat\libmf.lib) errno
sqrt	X	X	O	X	errno
sqrt	O	X	O	X	(libm.lib, softfloat\libm.lib) errno
sqrtl	X	X	O	X	errno
sqrtl	O	X	O	X	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib) errno
ceilf	O	X	X	O	
ceilf	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libmf.lib)
ceil	O	X	X	O	
ceil	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib)
ceill	O	X	X	O	
ceill	X	X	X	O	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib)
fabsf	X	X	X	O	
fabs	X	X	X	O	
fabsl	X	X	X	O	
floorf	O	X	X	O	
floorf	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libmf.lib)
floor	O	X	X	O	
floor	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib)
floorl	O	X	X	O	
floorl	X	X	X	O	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib)
roundf	O	X	X	O	
round	O	X	X	O	
roundl	O	X	X	O	
lroundf	O	O	O	X	errno

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
lround	O	O	O	X	errno
lroundl	O	O	O	X	errno
llroundf	O	O	O	X	errno
llround	O	O	O	X	errno
llroundl	O	O	O	X	errno
truncf	O	X	X	O	
trunc	O	X	X	O	
truncl	O	X	X	O	
fmodf	X	X	O	X	errno
fmodf	O	X	O	X	(softfloat\libmf.lib) errno
fmod	X	X	O	X	errno
fmod	O	X	O	X	(softfloat\libmf.lib) errno
fmodl	X	X	O	X	errno
copysignf	X	X	X	O	
copysign	X	X	X	O	
copysignl	X	X	X	O	
frexpf	X	X	O	X	errno
frexpf	O	X	O	X	(softfloat\libmf.lib) errno
frexp	X	X	O	X	errno
frexp	O	X	O	X	(softfloat\libmf.lib) errno
frexpl	X	X	O	X	errno
ldexpf	O	X	O	X	errno
ldexpf	X	X	O	X	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib) errno
ldexp	O	X	O	X	errno
ldexp	X	X	O	X	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib) errno
ldexpl	O	X	O	X	errno
ldexpl	X	X	O	X	(rhs8n.lib, rhs4n.lib, soft- float\libm.lib) errno
modff	O	X	X	O	

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
modff	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib)
modf	O	X	X	O	
modf	X	X	X	O	(rhs8n.lib, rhs4n.lib, softfloat\libm.lib)
modfl	O	X	X	O	
modfl	X	X	X	O	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib)
cosf	X	X	O	X	errno
cosf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
cos	O	X	O	X	errno
cos	X	X	O	X	(rhf4n.lib, rhf4z.lib) errno
cosl	O	X	O	X	errno
cosl	X	X	O	X	(rhf4n.lib, rhfnz.lib) errno
sinf	X	X	O	X	errno
sinf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
sin	O	X	O	X	errno
sin	X	X	O	X	(rhf4n.lib, rhf4z.lib) errno
sinl	O	X	O	X	errno
sinl	X	X	O	X	(rhf4n.lib, rhfnz.lib) errno
tanf	X	X	O	X	errno
tanf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
tan	X	X	O	X	errno
tan	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
tanl	X	X	O	X	errno
tanl	O	X	O	X	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib) errno
acosf	X	X	O	X	errno

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
acosf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
acos	X	X	O	X	errno
acos	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
acosl	X	X	O	X	errno
acosl	O	X	O	X	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib) errno
asinf	X	X	O	X	errno
asinf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
asin	X	X	O	X	errno
asin	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
asinl	X	X	O	X	errno
asinl	O	X	O	X	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib) errno
atanf	X	X	O	X	errno
atanf	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
atan	X	X	O	X	errno
atan	O	X	O	X	(rhs8n.lib, rhs4n.lib, libmf.lib, softfloat\libmf.lib) errno
atanl	X	X	O	X	errno
atanl	O	X	O	X	(rhs8n.lib, rhs4n.lib, soft-float\libm.lib) errno
atan2f	O	X	O	X	errno
atan2	O	X	O	X	errno
atan2l	O	X	O	X	errno
coshf	O	X	O	X	errno
cosh	O	X	O	X	errno
coshl	O	X	O	X	errno
sinhf	O	X	O	X	errno

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
sinh	O	X	O	X	errno
sinhl	O	X	O	X	errno
tanhf	O	X	O	X	errno
tanh	O	X	O	X	errno
tanhl	O	X	O	X	errno
fmax	X	X	X	O	
fmaxf	X	X	X	O	
fmaxl	X	X	X	O	
fmin	X	X	X	O	
fminf	X	X	X	O	
fminl	X	X	X	O	
calloc	X	O	O	X	Internal management data
free	X	O	O	X	Internal management data
malloc	X	O	O	X	Internal management data
realloc	X	O	O	X	Internal management data
_INITSCT_RH	X	X	X	O	
hdwinit	X	X	X	O	
_COM_fadd	X	X	X	O	
_COM_fsub	X	X	X	O	
_COM_fmul	X	X	X	O	
_COM_fdiv	X	X	X	O	
_COM_dadd	X	X	X	O	
_COM_dsub	X	X	X	O	
_COM_dmul	X	X	X	O	
_COM_ddiv	X	X	X	O	
_COM_mul64	X	X	X	O	
_COM_div64	X	X	X	O	
_COM_udiv64	X	X	X	O	
_COM_rem64	X	X	X	O	
_COM_urem64	X	X	X	O	
_COM_shll_64_32	X	X	X	O	
_COM_shrl_64_32	X	X	X	O	
_COM_shra_64_32	X	X	X	O	
_COM_neg64	X	X	X	O	

Function Name	Usage of .const	Usage of .data	Usage of .bss	Reen-trancy	Remarks (library for storage, cause of non-reentrancy)
_COM_itof	X	X	X	O	
_COM_itod	X	X	X	O	
_COM_utof	X	X	X	O	
_COM_utod	X	X	X	O	
_COM_i64tof	X	X	X	O	
_COM_i64tod	X	X	X	O	
_COM_u64tof	X	X	X	O	
_COM_u64tod	X	X	X	O	
_COM_ftoi	X	X	X	O	
_COM_dtoi	X	X	X	O	
_COM_ftou	X	X	X	O	
_COM_dtou	X	X	X	O	
_COM_ftoi64	X	X	X	O	
_COM_dtoi64	X	X	X	O	
_COM_ftou64	X	X	X	O	
_COM_dtou64	X	X	X	O	
_COM_ftod	X	X	X	O	
_COM_dtof	X	X	X	O	
_COM_fgt	X	X	X	O	
_COM_fge	X	X	X	O	
_COM_feq	X	X	X	O	
_COM_fne	X	X	X	O	
_COM_flt	X	X	X	O	
_COM_fle	X	X	X	O	
_COM_funord	X	X	X	O	
_COM_dgt	X	X	X	O	
_COM_dge	X	X	X	O	
_COM_deq	X	X	X	O	
_COM_dne	X	X	X	O	
_COM_dlt	X	X	X	O	
_COM_dle	X	X	X	O	
_COM_dunord	X	X	X	O	
__control_flow_integrity	O	X	X	O	Function list

8. STARTUP

This chapter explains the startup.

8.1 Outline

The startup process involves initializing sections for embedding the user application written in C language to the system or starting the main function.
This section assumes two types of programs: a program for a single core device, and a program for a multi-core device. The following shows examples of the configuration of the basic startup routine to run those programs.

Caution Additional processing other than that listed in this section is necessary depending on the specifications and restrictions on the device. For details, see the user's manual of the device.

8.2 Startup Routine

The startup routine is to be executed after the microcontroller has been reset and before the main function is executed. In the sample program, this routine is divided into two parts.

- Initialization routine for hardware
- Initialization routine for user program

8.2.1 Initialization routine for hardware

The initialization processing for hardware is configured by the following elements.

- RESET vector
- Interrupt handler table
- Exception handler routine
- Entry point
- Initialization of general-purpose registers
- Branch to initialization processing for each PE
- __exit routine
- Initialization processing for each PE
- Initialization processing for hardware
- Processing to make settings to use exception handler address of extended specifications (table lookup method)

In the sample program, these elements are located in boot.asm.

(1) RESET vector
A branch instruction to the entry point address of each processor element (PE) is allocated to the address where a branch of the program counter of each PE occurs when the microcontroller is reset. Since the RBASE register which holds the address of the RESET vector will hold a value in 512-byte units, the top of the RESET vector is aligned at the 512-byte boundary.
In the sample program, the RESET vector is located in the RESET_PE*n* section.

```
    .section "RESET_PE1", text
    .align   512
    jr32     __start ; RESET
    .align   16
    jr32     _Dummy  ; SYSERR
      :
    .align   16
    jr32     _Dummy_EI ; INTn(priority15)
```

The section name is optionally changeable, but it needs to be changed in conjunction with the -start option of the optimizing linker.

```
-start=RESET_PE1/01000000
```

Caution For the address of the RESET vector, see the user's manual of the device.

When the branch destination at the reset of each PE is the same address, a single RESET_PE*n* section is used in common by each PE.

(2) Interrupt handler table

When an exception handler address of the extended specifications (table lookup method) is used, the address of the exception handler routine in use is allocated to the corresponding element position in this table. Since the INTBP register which holds the table address will hold a value in 512-byte units, the top of the table is aligned at the 512-byte boundary.

In the sample program, the interrupt handler table is located in the EIINTTBL_PE*n* section.

```
    .section "EIINTTBL_PE1", const
    .align   512
    .dw      #_Dummy_EI ; INT0
    .dw      #_Dummy_EI ; INT1
    .dw      #_Dummy_EI ; INT2
    .rept    512 - 3
    .dw      #_Dummy_EI ; INTn
    .endm
```

Caution For the maximum number of elements in the table, see the user's manual of the device.

The section name is optionally changeable, but it needs to be changed in conjunction with the INTBP register setting in the startup.

```
    mov     #__sEIINTTBL_PE1, r10
    ldsr    r10, 4, 1       ; set INTBP
```

(3) Exception handler routine

This is a sample program for the exception handler routine of FE and EI levels. It repeats branches to itself without any operation.

Usually, it uses #pragma interrupt with the C source description.

```
    .align   2
_Dummy:
    br      _Dummy
_Dummy_EI:
    br      _Dummy_EI
```

(4) Entry point

The entry point is the label (address) to which the RESET vector branches at a reset.

```
    .align   2
    .public  __start
__start:
```

(5) Initialization of general-purpose registers

The general-purpose registers of each PE and the EIPC, CTPC, and FPEPC registers are initialized as a preparation to use the lockstep function.

```
    $nowarning
    mov     r0, r1
    $warning
    mov     r0, r2
    mov     r0, r3
     :
    mov     r0, r31
    ldsr    r0, 0, 0        ;  EIPC
    ldsr    r0, 16, 0       ;  CTPC
```

Since the FPEPC register can be initialized only after the FPU is enabled, it will be initialized later. See "Initial setting of FPU".

(6) Branch to initialization processing for each PE

This branch is executed commonly by each PE. The PE reads its processor element number (PEID) and branches from that value to the initialization processing prepared for each PE.

```
    stsr    0, r10, 2       ; get HTCFG0
    shr     16, r10         ; get PEID
    cmp     1, r10
    bz      .L.entry_PE1
    cmp     2, r10
    bz      .L.entry_PE2
     :
    cmp     7, r10
    bz      .L.entry_PE7
```

This processing is not required for a program for a single core device.

(7) __exit routine

The __exit routine repeats branches to itself to put PEs that are not in use to sleep.

```
__exit:
    br      __exit
```

(8) Initialization processing for each PE

This is to initiate the hardware initialization processing (_hdwinit_PE*n*) prepared for each PE and the processing to make settings to use the exception handler address of extended specifications (table lookup method) and then branch to the initialization routine (_cstart_pm*n*) for the user program.

```
.L.entry_PE1:
    jarl    _hdwinit_PE1, lp        ; initialize hardware
    mov     #__sEIINTTBL_PE1, r6
    jar     _set_table_reference_method, lp ; set table reference method
    jr32    __cstart_pm1
```

(9) Initialization processing for hardware

In the sample program, the RAM area is initialized as a preparation to use the ECC function. Global RAM and local RAM (for PE1) are initialized by the initialization processing for PE1 and local RAM (for PE2) is initialized by the initialization processing for PE2.

```
    .align  2
_hdwinit_PE1:
    mov     lp, r29                 ; save return address
    ; clear Global RAM
    mov     GLOBAL_RAM_ADDR, r6
    mov     GLOBAL_RAM_END, r7
    jarl    _zeroclr4, lp
    ; clear Local RAM PE1
    mov     LOCAL_RAM_PE1_ADDR, r6
    mov     LOCAL_RAM_PE1_END, r7
    jarl    _zeroclr4, lp
    mov     r29, lp
    jmp     [lp]

    .align  2
_hdwinit_PE2:
    mov     lp, r29                 ; save return address
    ; clear Local RAM PE2
    mov     LOCAL_RAM_PE2_ADDR, r6
    mov     LOCAL_RAM_PE2_END, r7
    jarl    _zeroclr4, lp
    mov     r29, lp
    jmp     [lp]

    .align  2
_zeroclr4:
    br      .L.zeroclr4.2
.L.zeroclr4.1:
    st.w    r0, [r6]
    add     4, r6
.L.zeroclr4.2:
    cmp     r6, r7
    bh      .L.zeroclr4.1
    jmp     [lp]
```

Caution In the sample program, an invalid address is specified with a macro. For the RAM addresses to be initialized, see the user's manual of the device.

(10) Processing to make settings to use exception handler address of extended specifications (table lookup method)
This processing sets the address of the interrupt handler table to the INTBP register and sets the interrupt control registers.

```
    .align  2
_set_table_reference_method:
    ldsr    r6, 4, 1        ; set INTBP
    mov     ICBASE, r10     ; get interrupt control register address
    set1    6, 0[r10]       ; set INT0 as table reference
    set1    6, 2[r10]       ; set INT1 as table reference
    set1    6, 4[r10]       ; set INT2 as table reference
    jmp     [lp]
```

8.2.2 Initialization routines of user programs

The initialization processing for the user program is configured by the following elements.

- Stack area
- Entry point
- Initialization of base registers
- Initialization of RAM sections
- Initial setting of FPU

- Initial setting of exception handling
- Branch to main function

In the sample program, these elements are located in cstart.asm.

(1) Stack area

The stack area is used for code generated by the compiler. The CC-RH generates code based on the assumption that the stack pointer (sp) is allocated at the 4-byte boundary and the stack area is made to be grown in the direction of address 0x0. Therefore, an address aligned at the 4-byte boundary on the 0xffffffff address side of the .stack.bss section needs to be specified for the stack pointer (sp).

```
STACKSIZE    .set    0x200
    .section ".stack.bss", bss
    .align   4
    .ds      (STACKSIZE)
    .align   4
_stacktop:
```

(2) Entry point

The entry point is the label (address) to which the hardware initialization routine branches at a reset.

```
    .public __cstart_pm1
    .align  2
__cstart_pm1:
```

(3) Initialization of base registers

The stack pointer, gp register, and ep register are all initialized.

```
    mov     #_stacktop, sp      ;  set sp register
    mov     #__gp_data, gp      ;  set gp register
    mov     #__ep_data, ep      ;  set ep register
```

For details on __gp_data and __ep_data, see "8.4 Symbols".

(4) Initialization of RAM sections

Variable areas defined in the C source program or assembly source program are initialized. Prepare a table that holds the addresses of sections to be initialized and pass the address of the table to the _INITSCT_RH library function and call it.

The initialization table for initialized data sections should be written in the following format.

```
    .section ".INIT_DSEC.const", const
    .align   4
    .dw    #__s<section name 1>, #__e<section name 1>,
                  #__s<name of the corresponding RAM section to be initialized>
    .dw    #__s<section name 2>, #__e<section name 2>,
                  #__s<name of the corresponding RAM section to be initialized>
     :
    .dw    #__s<section name n>, #__e<section name n>,
                  #__s<name of the corresponding RAM section to be initialized>
```

Use the -rom option of the optimizing linker to specify the RAM sections to be initialized.

```
-rom=.data=.data.R
-rom=.sdata=.sdata.R
```

In this case, code of the initialization table is as follows:

```
    .section ".INIT_DSEC.const", const
    .align   4
    .dw      #__s.data,  #__e.data,  #__s.data.R
    .dw      #__s.sdata, #__e.sdata, #__s.sdata.R
```

The initialization table for uninitialized data sections should be written in the following format.

```
    .section ".INIT_BSEC.const", const
    .align   4
    .dw      #__s<section name 1>, #__e<section name 1>
    .dw      #__s<section name 2>, #__e<section name 2>
     :
    .dw      #__s<section name n>, #__e<section name n>
```

Use the -start option of the optimizing linker to specify the addresses of ROM sections containing initial values and the corresponding RAM sections to be initialized.

```
-start=.data,.sdata/00008000
-start=.data.R,.sdata.R/fedf0000
-start=.bss.sbss/fedf8000
```

Note Allocate sections to ROM and RAM in accord with the memory map of the target MCU. For details, refer to the user's manual for the MCU.

The start address and end address of the initialization table are passed through parameters of the _INITSCT_RH function and initialization is executed.
The start of each initialization table must be aligned at the 4-byte boundary.

```
    mov     #__s.INIT_DSEC.const, r6
    mov     #__e.INIT_DSEC.const, r7
    mov     #__s.INIT_BSEC.const, r8
    mov     #__e.INIT_BSEC.const, r9
    jarl32  __INITSCT_RH, lp     ; initialize RAM area
```

For the usage method of _INITSCT_RH, see "7.4.11 RAM section initialization function".

(5) Initial setting of FPU
Reference the PID register to confirm whether the FPU is available.
If available, make the initial settings.
Set the PSW.CU0 bit to 1 to enable usage of the FPU.
Set the FPU operating mode for the FPSR register.
The FPEPC register is initialized as a preparation to use the lockstep function.

```
    stsr    6, r10, 1        ; r10 <- PID
    shl     21, r10
    shr     30, r10
    bz      .L1              ; detect FPU
    stsr    5, r10, 0        ; r10 <- PSW
    movhi   0x0001, r0, r11
    or      r11, r10
    ldsr    r10, 5, 0        ; enable FPU
    movhi   0x0002, r0, r11
    ldsr    r11, 6, 0        ; initialize FPSR
    ldsr    r0, 7, 0         ; initialize FPEPC
.L1:
```

Delete these codes from programs that do not use the FPU.

(6) Initial setting of exception handling
Set the PSW.ID bit to 0 to enable occurrence of exceptions.
Set the PSW.UM bit to 1 to enter the user mode.
To reflect the settings as soon as a branch to the main function occurs, write the settings to the FEPSW instead of the PSW; the FEPSW settings are reflected to the PSW when the feret instruction is executed.

```
    stsr    5, r10, 0        ; r10 <- PSW
    xori    0x0020, r10, r10 ; enable interrupt
    movhi   0x4000, r0, r11
    or      r11, r10         ; supervisor mode -> user mode
    ldsr    r10, 3, 0        ; FEPSW <- r10
```

(7) Branch to main function
A branch to the main function occurs when the feret instruction is executed with the address of the main function set to the FEPC register and the address of the _exit function to be executed subsequent to the main function set to the r31 register.

```
    mov     #_exit, lp       ; lp <- #_exit
    mov     #_main, r10
    ldsr    r10, 2, 0        ; FEPC <- #_main
    feret
```

8.2.3 Passing information between projects

When a program for a multi-core device is configured of a single boot loader project and multiple application projects, the -fsymbol option of the optimizing linker is used to reference information of application programs from the boot loader program.

If an application project is linked with the -fsymbol option specified in the optimizing linker, the name and address value of the public label existing in the section specified by the option are output to the symbol address file (.fsy file). Information can be referenced by specifying the symbol address file that is output from each application project as input to the boot loader project.

An example of passing information is shown below.

```
;; cstart.asm
    .section  ".text.cmn", text
    .public   __cstart_pm1       ; public is specified to output information to .fsy.
__cstart_pm1:
```

When the application project is linked, the -fsymbol option specifies the .text.cmn section where the __cstart_pm1 label exists. In this case, the pm1.fsy file is generated.

```
> rlink cstart.obj -output=pm1.abs -fsymbol=.text.cmn
```

The boot loader project references the label on the application project side.

```
;; boot.asm
    jr32     #__cstart_pm1
```

When the boot loader project is compiled, by inputting the .fsy file generated from each application project, the address values in .fsy files resolve references to labels.

```
> ccrh boot.asm pm1.fsy pm2.fsy -oboot.abs
```

Caution Labels output in .fsy files are handled as public labels also in the boot loader project. Therefore, if labels with the same name have been output to .fsy files from multiple application projects or a label with the same name has been defined in the boot loader project, a multiple definition error will occur at linking of the boot loader project.

8.3 Coding Example

Examples of boot.asm and cstart.asm are shown below.
boot.asm

```
    ; if using eiint as table reference method,
    ; enable next line's macro.

    ;USE_TABLE_REFERENCE_METHOD .set 1

;---------------------------------------------------------------------------
;   exception vector table
;---------------------------------------------------------------------------
    .section "RESET_PE1", text
    .align   512
    jr32     __start   ; RESET

    .align   16
    jr32     _Dummy    ; SYSERR

    .align   16
    jr32     _Dummy

    .align   16
    jr32     _Dummy    ; FETRAP

    .align   16
    jr32     _Dummy_EI ; TRAP0

    .align   16
    jr32     _Dummy_EI ; TRAP1

    .align   16
    jr32     _Dummy    ; RIE

    .align   16
    jr32     _Dummy_EI ; FPP/FPI

    .align   16
    jr32     _Dummy    ; UCPOP

    .align   16
    jr32     _Dummy    ; MIP/MDP

    .align   16
    jr32     _Dummy    ; PIE

    .align   16
    jr32     _Dummy

    .align   16
    jr32     _Dummy    ; MAE

    .align   16
    jr32     _Dummy

    .align   16
    jr32     _Dummy    ; FENMI

    .align   16
    jr32     _Dummy    ; FEINT
```

```
    .align   16
    jr32     _Dummy_EI ; INTn(priority0)

    .align   16
    jr32     _Dummy_EI ; INTn(priority1)

    .align   16
    jr32     _Dummy_EI ; INTn(priority2)

    .align   16
    jr32     _Dummy_EI ; INTn(priority3)

    .align   16
    jr32     _Dummy_EI ; INTn(priority4)

    .align   16
    jr32     _Dummy_EI ; INTn(priority5)

    .align   16
    jr32     _Dummy_EI ; INTn(priority6)

    .align   16
    jr32     _Dummy_EI ; INTn(priority7)

    .align   16
    jr32     _Dummy_EI ; INTn(priority8)

    .align   16
    jr32     _Dummy_EI ; INTn(priority9)

    .align   16
    jr32     _Dummy_EI ; INTn(priority10)

    .align   16
    jr32     _Dummy_EI ; INTn(priority11)

    .align   16
    jr32     _Dummy_EI ; INTn(priority12)

    .align   16
    jr32     _Dummy_EI ; INTn(priority13)

    .align   16
    jr32     _Dummy_EI ; INTn(priority14)

    .align   16
    jr32     _Dummy_EI ; INTn(priority15)

    .section "EIINTTBL_PE1", const
    .align   512
    .dw      #_Dummy_EI ; INT0
    .dw      #_Dummy_EI ; INT1
    .dw      #_Dummy_EI ; INT2
    .rept    512 - 3
    .dw      #_Dummy_EI ; INTn
    .endm

    .section ".text", text
    .align   2
_Dummy:
    br       _Dummy
```

```
_Dummy_EI:
    br      _Dummy_EI

;-------------------------------------------------------------------------------
;   startup
;-------------------------------------------------------------------------------
    .section ".text", text
    .align   2
    .public  __start
__start:
$if 1    ; initialize register
    $nowarning
    mov     r0, r1
    $warning
    mov     r0, r2
    mov     r0, r3
    mov     r0, r4
    mov     r0, r5
    mov     r0, r6
    mov     r0, r7
    mov     r0, r8
    mov     r0, r9
    mov     r0, r10
    mov     r0, r11
    mov     r0, r12
    mov     r0, r13
    mov     r0, r14
    mov     r0, r15
    mov     r0, r16
    mov     r0, r17
    mov     r0, r18
    mov     r0, r19
    mov     r0, r20
    mov     r0, r21
    mov     r0, r22
    mov     r0, r23
    mov     r0, r24
    mov     r0, r25
    mov     r0, r26
    mov     r0, r27
    mov     r0, r28
    mov     r0, r29
    mov     r0, r30
    mov     r0, r31
    ldsr    r0, 0, 0        ;  EIPC
    ldsr    r0, 16, 0       ;  CTPC
$endif

$if 1
    ; jump to entry point of each PE
    stsr    0, r10, 2       ; get HTCFG0
    shr     16, r10         ; get PEID

    cmp     1, r10
    bz      .L.entry_PE1
    cmp     2, r10
    bz      .L.entry_PE2
    cmp     3, r10
    bz      .L.entry_PE3
    cmp     4, r10
    bz      .L.entry_PE4
```

```
    cmp      5, r10
    bz       .L.entry_PE5
    cmp      6, r10
    bz       .L.entry_PE6
    cmp      7, r10
    bz       .L.entry_PE7
__exit:
    br       __exit

.L.entry_PE1:
    jarl     _hdwinit_PE1, lp ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
    mov      #__sEIINTTBL_PE1, r6
    jarl     _set_table_reference_method, lp ; set table reference method
$endif

    jr32     __cstart_pm1

.L.entry_PE2:
    jarl     _hdwinit_PE2, lp ; initialize hardware
;$ifdef USE_TABLE_REFERENCE_METHOD
;   mov      #__sEIINTTBL_PE2, r6
;   jarl     _set_table_reference_method, lp ; set table reference method
;$endif
;   jr32     __cstart_pm2
    br       __exit

.L.entry_PE3:
    br       __exit
.L.entry_PE4:
    br       __exit
.L.entry_PE5:
    br       __exit
.L.entry_PE6:
    br       __exit
.L.entry_PE7:
    br       __exit
$endif

;-----------------------------------------------------------------------------
; hdwinit_PE1
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------------
    GLOBAL_RAM_ADDR     .set    0
    GLOBAL_RAM_END      .set    0
    LOCAL_RAM_PE1_ADDR  .set    0
    LOCAL_RAM_PE1_END   .set    0

    .align    2
_hdwinit_PE1:
    mov      lp, r29           ; save return address

    ; clear Global RAM
    mov      GLOBAL_RAM_ADDR, r6
    mov      GLOBAL_RAM_END, r7
    jarl     _zeroclr4, lp

    ; clear Local RAM PE1
    mov      LOCAL_RAM_PE1_ADDR, r6
    mov      LOCAL_RAM_PE1_END, r7
    jarl     _zeroclr4, lp
```

```
    mov       r29, lp
    jmp       [lp]

;-------------------------------------------------------------------------------
; hdwinit_PE2
; Specify RAM addresses suitable to your system if needed.
;-------------------------------------------------------------------------------
    LOCAL_RAM_PE2_ADDR  .set    0
    LOCAL_RAM_PE2_END   .set    0

    .align    2
_hdwinit_PE2:
    mov       lp, r14           ; save return address

    ; clear Local RAM PE2
    mov       LOCAL_RAM_PE2_ADDR, r6
    mov       LOCAL_RAM_PE2_END, r7
    jarl      _zeroclr4, lp

    mov       r14, lp
    jmp       [lp]

;-------------------------------------------------------------------------------
;   zeroclr4
;-------------------------------------------------------------------------------
    .align    2
_zeroclr4:
    br        .L.zeroclr4.2
.L.zeroclr4.1:
    st.w      r0, [r6]
    add       4, r6
.L.zeroclr4.2:
    cmp       r6, r7
    bh        .L.zeroclr4.1
    jmp       [lp]

$ifdef USE_TABLE_REFERENCE_METHOD
;-------------------------------------------------------------------------------
;   set table reference method
;-------------------------------------------------------------------------------
    ; interrupt control register address
    ICBASE    .set 0xfffeea00

    .align    2
_set_table_reference_method:
    ldsr      r6, 4, 1      ; set INTBP

    ; Some interrupt channels use the table reference method.
    mov       ICBASE, r10   ; get interrupt control register address
    set1      6, 0[r10]     ; set INT0 as table reference
    set1      6, 2[r10]     ; set INT1 as table reference
    set1      6, 4[r10]     ; set INT2 as table reference

    jmp       [lp]
$endif
;-------------------- end of start up module ------------------;
```

cstart.asm

```
;-------------------------------------------------------------------------------
;   system stack
;-------------------------------------------------------------------------------
STACKSIZE    .set    0x200
    .section ".stack.bss", bss
    .align   4
    .ds      (STACKSIZE)
    .align   4
_stacktop:

;-------------------------------------------------------------------------------
;   section initialize table
;-------------------------------------------------------------------------------
    .section ".INIT_DSEC.const", const
    .align   4
    .dw      #__s.data, #__e.data, #__s.data.R

    .section ".INIT_BSEC.const", const
    .align   4
    .dw      #__s.bss, #__e.bss

;-------------------------------------------------------------------------------
;   startup
;-------------------------------------------------------------------------------
    .section ".text.cmn", text
    .public  __cstart_pm1
    .align   2
__cstart_pm1:
    mov      #_stacktop, sp     ;  set sp register
    mov      #__gp_data, gp     ;  set gp register
    mov      #__ep_data, ep     ;  set ep register

    mov      #__s.INIT_DSEC.const, r6
    mov      #__e.INIT_DSEC.const, r7
    mov      #__s.INIT_BSEC.const, r8
    mov      #__e.INIT_BSEC.const, r9
    jarl32   __INITSCT_RH, lp   ;  initialize RAM area

    ; set various flags to PSW via FEPSW

    stsr     5, r10, 0          ; r10 <- PSW

    movhi    0x0001, r0, r11
    or       r11, r10
    ldsr     r10, 5, 0          ; enable FPU

    movhi    0x0002, r0, r11
    ldsr     r11, 6, 0          ; initialize FPSR
    ldsr     r0, 7, 0           ; initialize FPEPC

    stsr     5, r10, 0          ; r10 <- PSW

    ;xori    0x0020, r10, r10   ; enable interrupt

    ;movhi   0x4000, r0, r11
    ;or      r11, r10           ; supervisor mode -> user mode

    ldsr     r10, 3, 0          ; FEPSW <- r10
```

```
    mov      #_exit, lp          ; lp <- #_exit
    mov      #_main, r10
    ldsr     r10, 2, 0           ; FEPC <- #_main

    ; apply PSW and PC to start user mode
    feret

_exit:
    br       _exit               ;  end of program

;-------------------------------------------------------------------------------
;   dummy section
;-------------------------------------------------------------------------------
    .section    ".data", data
.L.dummy.data:
    .section    ".bss", bss
.L.dummy.bss:
    .section    ".const", const
.L.dummy.const:
    .section    ".text", text
.L.dummy.text:
;------------------- end of start up module -------------------;
```

8.4 Symbols

In the CC-RH, the following symbols are used as necessary.

- __gp_data symbol
  It is the value to be set in the global pointer register (r4).
  It is used to reference variables allocated to the sdata or sdata23 attribute section with short instructions. It is also used by the position independent data (PID) facility.
- __ep_data symbol
  It is the value to be set in the element pointer register (r30).
  It is used to reference variables allocated to the tdata, edata, or edata23 attribute section with short instructions. It is also used by the position independent data (PID) facility.
- __pc_data symbol
  It is used to reference variables allocated to the pcconst16, pcconst23, or pcconst32 attribute section.

The method for determining each symbol value is described here.

8.4.1 __gp_data

The value of __gp_data is determined in the following order of precedence.

(1) When __gp_data is defined in the application, that value is used.

Note The compiler generates code based on the assumption that __gp_data is aligned at the 2-byte boundary. Therefore, make sure the defined value is a multiple of 2 when defining __gp_data in the application.

(2) When there is only a reference to __gp_data in the application, the optimizing linker (rlink) automatically determines the __gp_data value in the following order of precedence.

(2-1) If there is an sdata or sbss attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.

(2-2) If there is an sdata23 or sbss23 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.

(2-3) If there is an sdata32 or sbss32 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.

(2-4) If none of the above sections exist and there is only a reference to __gp_data, the value is 0.

Note, however, that if the value attempted to be defined is an odd value, 1 is added to that value.
The compiler will not generate code to directly reference __gp_data. Normally, a reference to __gp_data indicates the processing to set the __gp_data value to the global pointer register (r4) in the startup routine.

(3) When there is no definition or reference regarding __gp_data in the application, the optimizing linker (rlink) does not generate __gp_data. If there is code to reference a GP-relative section in this state, an error will occur at linkage.

Undefined external symbol "GP-symbol (__gp_data)" referenced in "FILE"

8.4.2 __ep_data

The value of __ep_data is determined in the following order of precedence.

(1) When __ep_data is defined in the application, that value is used.

Note The compiler generates code based on the assumption that __ep_data is aligned at the 2-byte or 4-byte boundary. Therefore, make sure the defined value is a multiple of 4 when defining __ep_data in the application.

(2) When there is only a reference to __ep_data in the application, the optimizing linker (rlink) automatically determines the __ep_data value in the following order of precedence.

(2-1) If there is a tdata, tdata4, tbss4, tdata5, tbss5, tdata7, tbss7, tdata8, or tbss8 attribute section, the value is the minimum address of all of those sections that have that attribute in the following order of precedence.
(a) tdata attribute section
(b) tdata4 or tbss4 attribute section
(c) tdata5 or tbss5 attribute section
(d) tdata7 or tbss7 attribute section
(e) tdata8 or tbss8 attribute section

Note If sections with the above attributes are used, they should be allocated in the above order. The sld or sst instruction has an unsigned offset. Therefore, if a low-priority section is allocated to an address that is smaller than that for a high-priority section, the low-priority section cannot be referenced with the sld or sst instruction. In this case, an error will occur at linkage.

(2-2) If there is an edata or ebss attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.
(2-3) If there is an edata23 or ebss23 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.
(2-4) If there is an edata32 or ebss32 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.
(2-5) If none of the above sections exist and there is only a reference to __ep_data, the value is 0.

Note, however, that if the value attempted to be defined is an odd value, 1 is added to that value.
The compiler will not generate code to directly reference __ep_data. Normally, a reference to __ep_data indicates the processing to set the __ep_data value to the element pointer register (r30) in the startup routine.

(3) When there is no definition or reference regarding __ep_data in the application, the optimizing linker (rlink) does not generate __ep_data. If there is code to reference an EP-relative section in this state, an error will occur at linkage.

Undefined external symbol "EP-symbol (__ep_data)" referenced in "FILE"

8.4.3 __pc_data

The value of __pc_data is determined in the following order of precedence.

(1) When __pc_data is defined in the application, that value is used.

Note The compiler generates code based on the assumption that __pc_data is aligned at the 2-byte boundary. Therefore, make sure the defined value is a multiple of 2 when defining __pc_data in the application.

(2) When there is only a reference to __pc_data in the application, the optimizing linker (rlink) automatically determines the __pc_data value in the following order of precedence.
(2-1) If there is a pcconst16 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.
(2-2) If there is a pcconst23 attribute section, the value is the intermediate value of the minimum address and maximum address of all of those sections.
(2-3) If there is a pcconst32 attribute section, the value is the minimum address of all of those sections.
(2-4) If none of the above sections exist and there is only a reference to __pc_data, the value is 0.

Note, however, that if the value attempted to be defined is an odd value, 1 is added to that value.

8.5 Creating ROM Images

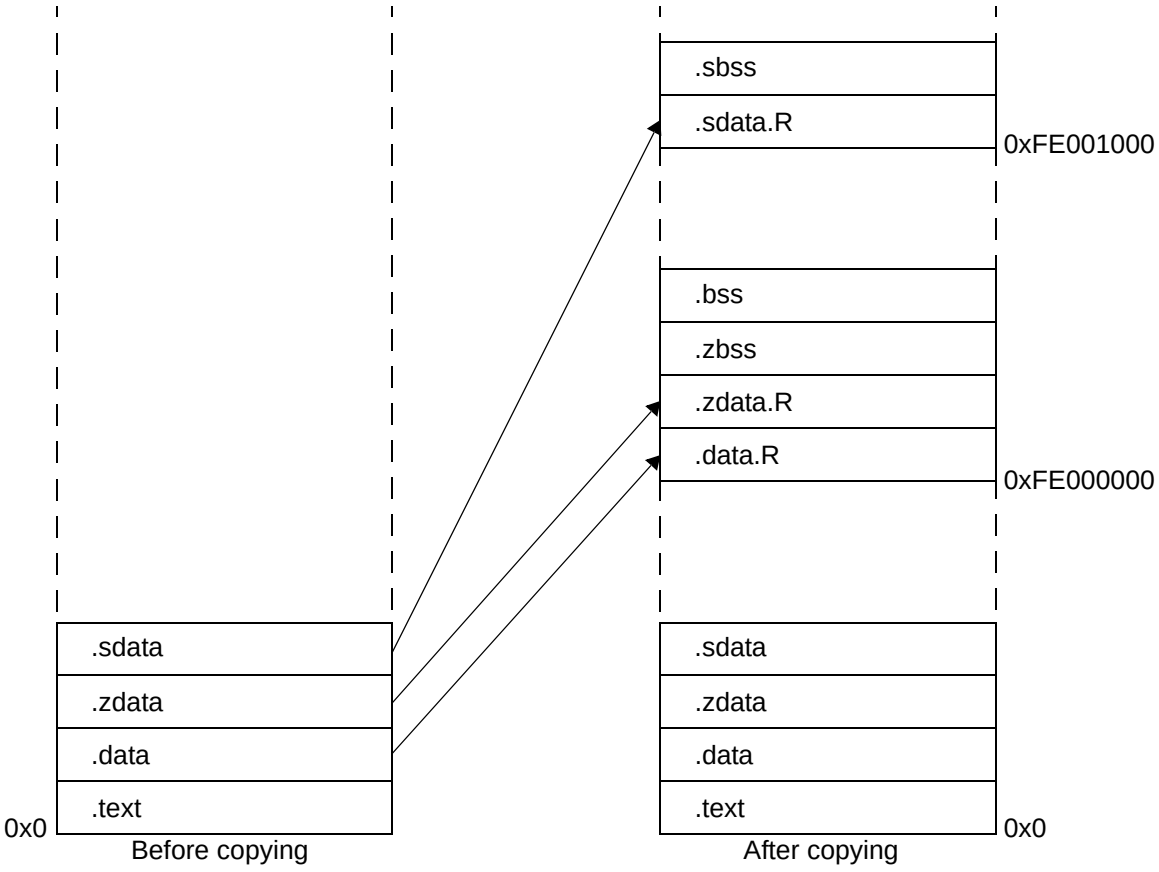
This section gives an outline of the creation of ROM images that are required for embedded applications.

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-RH 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.

For details on the creation of ROM images and copying them to RAM, refer to (4), Initialization of RAM sections, in section 8.2.2, Initialization routines of user programs.

The following figure shows the copying of initial values from ROM to RAM.

Figure 8.1 Memory Maps before and after the Copying of Initial Values



Remarks

- As well as the initial values of variables, you can copy program code that you wish to run from RAM.
- When you are using an in-circuit emulator for debugging, for example, you can also download load modules directly to ROM or RAM and execute them from the given types of memory. In such cases, the copying of initial values is not necessary.

8.6 PIC/PID Facility

The method for creating a program that can allocate functions or data to desired positions in memory (position-independent program) is described in this section.

In the CC-RH, the objects to be allocated and the facilities are classified into the following three types.

- A facility that can allocate code (function) to a given address and execute it is called the PIC (Position Independent Code) facility.
- A facility that can allocate the constant data (const variable) to a given address and reference it is called the PIROD (Position Independent Read Only Data) facility.
- A facility that can allocate data (variable) to a given address and reference it is called the PID (Position Independent Data) facility.

8.6.1 PIC

Position-independent is achieved by referencing the code in PC-relative mode in the CC-RH.

When the -pic option is specified, the default section to which code is allocated will be the .pctext section. Execution at a given address is enabled by calling functions that were allocated to the .pctext section or referencing addresses in PC-relative mode.

8.6.2 PIROD

Position-independent is achieved by referencing the constant data in PC-relative mode in the CC-RH. Since this is the same relative reference method as that for code, the PIROD facility has to be enabled simultaneously with the PIC facility. Then, the constant data has to be appropriately allocated to a desirable position for execution so that the distance between the code area and constant data area is the same as that at linkage.

When the -pirod option is specified, the default section to which the constant data is allocated will be the .pcconst32 section. Allocation and reference to a given position is enabled by referencing the constant data that was allocated to the .pcconst32 section in PC-relative mode. The allocation section can be changed to the .pcconst16 section or .pcconst23 section by using the #pragma section directive or -Xsection option.

8.6.3 PID

Position-independent is achieved by referencing data in GP-relative or EP-relative mode in the CC-RH.

When the -pid option is specified, the default section to which data is allocated will be the .sdata32 section and .sbss32 section. The code to reference data allocated to these sections is output in GP-relative mode. Similarly as to when not using the PID facility, the allocation section can be changed to the .sdata section or .sdata23 section by using the #pragma section directive or -Xsection option.

8.6.4 Referencing from a position-independent program to a position-dependent program

Position-dependent programs, such as existing resources, can be referenced from position-independent programs by following a certain procedure. The position-dependent programs in this case are called the common part.

(1) The common part must already be created before creating position-independent programs as a precondition. When creating the executable form of the common part, the addresses of the functions or variables to be referenced from position-independent programs are output to an .fsy file, using the -fsymbol option of the optimizing linker.

- To reference library functions not from within the common part but from position-independent programs as the common part, write a pseudo reference code such as that shown below to link the library functions to the common part.

```
#include <string.h>
void* const dummy_libcall[] = {&memcpy, &memcmp, &strcpy};
```

The common.fsy file is output with the following manipulation.

```
>ccrh dummy_libcall.c -ocommon.abs -Xlk_option=-fsymbol=.text
```

To reference from position-independent programs, the executable form of the common part that has already been created, confirm the addresses of the functions or variables to be referenced in the link map file, etc. of the common part and write them to an .fsy file.

Example When addresses are displayed in the link map file as shown below:

```
FILE=memcmp
                                00002000  00002023        24
  _memcmp
                                00002000          0   none ,g          *
FILE=memcpy
                                00002024  0000203b        18
  _memcpy
                                00002024          0   none ,g          *
FILE=strcpy
                                0000203c  0000204f        14
  _strcpy
                                0000203c          0   none ,g          *
```

Code to be written to the common.fsy file:

```
        .public _memcmp
_memcmp .equ 0x2000
        .public _memcpy
_memcpy .equ 0x2024
        .public _strcpy
_strcpy .equ 0x203c
```

(2) When creating a position-independent program, write a declaration of the functions or variables in the common part to be referenced and a proccessing to reference them. In this case, the section in which the declaration belongs should comply with the section containing the definition of functions or variables on the common part side. The PIC functions on the side to perform reference should be defined in the section for PIC.

Note Even when the -pid, -pirod, or -pid option is specified, a section relocation attribute that is position-dependent (e.g. text, const, and r0_disp16) can be specified in a #pragma section directive. Only the declaration of functions or variables can be written in this case. When the definition of functions or variables is written, an error will occur.

```
#pragma section text
extern  void *memcpy(void *, const void *, unsigned long);
extern  int memcmp(const void *, const void *, size_t);
extern  char *strcpy(char *, const char *);

#pragma section pctext  /* When defining a PIC function, the section relocation
attribute must be returned to that for PIC. */

void pic_func(char* a, char* b, unsigned long c) {
  memcpy(a, b, c);
}
```

Write code to reference functions or variables in the common part and build a position-independent program. By building this program together with the .fsy file that was generated in the common part, reference to functions or variables on the common part side can be achieved by absolute addressing.

```
>ccrh pic.c common.fsy
```

(3) There is a restriction on the method for referencing the functions or variables in the common part which can be referenced from position-independent programs.
The table below shows whether reference is possible and the reference method for a case between position-independent programs or a case between a position-independent program and the common part.

		Reference destination					
		PIC function	Non-PIC function	PIROD variable	Non-PIROD variable	PID variable[Note 2]	Non-PID variable
Reference source	PIC function	PC-relative	R0-relative	PC-relative	R0-relative	GP- or EP-relative	GP- or EP-relative R0-relative
	Non-PIC function	Not possible[Note 1]	PC-relative R0-relative	Not possible[Note 1]	R0-relative	GP- or EP-relative	GP- or EP-relative R0-relative

Note 1. When a non-PIC function is linked, no kind of direct reference is possible because the linker cannot identify the addresses of PIC functions or PIROD functions at execution. A possible method of reference is to receive the pointer at execution and perform reference via the pointer.

Note 2. PID variables indicate the variables that were compiled with the -pid option specified instead of the variables in general which are allocated to the GP-relative or EP-relative sections.

8.6.5 Restrictions on PIC/PID facility

(1) Position-independent code or data operates at addresses different from those at linkage. Therefore, the addresses of position-independent code or data cannot be specified as initializers of static variables.

(2) The GP-relative or EP-relative sections can be used for both position-independent data and position-dependent data. However, since the GP and EP registers are shared, if the GP or EP register value is changed to use the PID facility, the reference addresses of position-dependent data are also changed. It is recommended to unify in the entire program whether the GP or EP register is each used for position-independent data or position-dependent data.

(3) The standard library does not support the PIC, PIROD, and PID facilities. The facilities should be positioned and used in the common part.

(4) Since the section name is changed when the PIC or PID facility is used, the -start option specification of the linker also needs to be changed.

(5) The standard startup routine cannot be used when the PIC or PID facility is used. Create the startup routine with reference to "8.6.6 Startup routine".

8.6.6 Startup routine

When the PIC, PIROD, or PID facility is used, the following processes in the startup routine need to be changed.

- Reset vector
- Initialization of base registers
- Initialization of RAM sections
- Branch to main function

(1) Reset vector
When the entire program is to be configured as position-independent, it is necessary to jump from the reset vector to a desired position. Therefore, the branch destination address should be written to a specific RAM area or data flash area as an example. To restart the program without shutting off the power supply of the microcontroller, the branch destination address should be stored in a specific register.

In the reset vector, the address to be jumped to is acquired and a register indirect branch is executed.

```
    cstart_address .set 0xXXXXXXXX    ; Address storing the branch destination
                                      ; address to be executed

    .section "RESET", text
    .align   512
    mov cstart_address, r10
    ld.w 0[r10], r10
    jmp [r10]
```

(2) Initialization of base registers
When the PID facility is used, first the means of passing the offset information (hereafter referred to as the RAM offset value), such as how much to shift the allocation position at execution from the start address of the RAM section that was specified at linkage, has to be decided in advance. For example, the RAM offset value should be written to a specific RAM area or data flash area.[Note]

Note Since the specific area has to be referenced with an absolute address in this case, the PID or PIROD facility is not supported.

To restart the program without shutting off the power supply of the microcontroller, the RAM offset value should be stored in a specific register.
The RAM offset value that was received is added to the base register, and this address will be used as the base address at execution.

```
    mov     0xfedf0000, r28       ; Memory address for passing RAM offset value
    ld.w    0[r28], r28           ; Offset (RAM offset) between data arrangement
                                  ; at linkage and data arrangement at execution

    mov     #_stacktop, sp        ; set sp register
    mov     #__gp_data, gp        ; set gp register
    mov     #__ep_data, ep        ; set ep register
    add     r28, sp
    add     r28, gp
    add     r28, ep
```

(3) Initialization of RAM sections
The _INITSCT_RH() function cannot be used for initializing sections when the PID facility is used because the section information table is only for input. Due to this, the initial values are to be directly copied in the startup routine.

The offset between the allocation address of the code area or constant data area at linkage and execution is obtained as an advance preparation. Hereafter, this offset is referred to as the ROM offset value.

```
    jarl     .pic_base, r29   ; Address of .pic_base label at execution is stored
                              ; in r29
.pic_base:
    mov      #.pic_base, r10  ; Address of .pic_base label at linkage is stored
                              ; in r10
    sub      r10, r29         ; Value obtained by subtracting r10 from r29 is the
                              ; ROM offset value
```

Next, initialization of sections with initial value is performed.
For a section to be initialized, the start and end addresses of the copy source of the initial value and the copy destination address are stored in the r6, r7, and r8 registers, respectively.

```
    mov      #__s.sdata32, r6
    mov      #__e.sdata32, r7
    mov      #__s.sdata32.R, r8
```

When the PIROD facility is used, the ROM offset value is added to the start address (r6 register) and end address (r7 register) of the copy source of the initial value.

```
    add      r29, r6
    add      r29, r7
```

When the PID facility is used, the RAM offset value is added to the address to which data is copied (r8 register).

```
    add      r28, r8
```

The copy routine is called here because the preparation for copy is completed.

```
    jarl     _copy4, lp
....
    ; r6: source begin  (4-byte aligned)
    ; r7: source end  (r6 <= r7)
    ; r8: destination begin  (4-byte aligned)
    .align    2
_copy4:
    sub      r6, r7
.copy4.1:
    cmp      4, r7
    bl       .copy4.2
    ld.w     0[r6], r10
    st.w     r10, 0[r8]
    add      4, r6
    add      4, r8
    add      -4, r7
    br       .copy4.1
.copy4.2:
    cmp      2, r7
    bl       .copy4.3
    ld.h     0[r6], r10
    st.h     r10, 0[r8]
    add      2, r6
    add      2, r8
    add      -2, r7
.copy4.3:
    cmp      0, r7
    bz       .copy4.4
    ld.b     0[r6], r10
    st.b     r10, 0[r8]
.copy4.4:
    jmp      [lp]
```

Processing up to this point is repeated for the number of sections that require an initial value.

Next, sections with no initial value are initialized with 0. The start address and end address of a target section are stored in the r6 and r7 registers, respectively.

```
    mov      #__s.sbss32, r6
    mov      #__e.sbss32, r7
```

When the PID facility is used, the RAM offset value is added to the start address (r6 register) and end address (r7 register).

```
    add      r28, r6
    add      r28, r7
```

The initialization routine is called and the target sections are initialized with 0.

```
    jarl     _clear4, lp
....
    ; r6: destination begin (4-byte aligned)
    ; r7: destination end  (r6 <= r7)
    .align   2
_clear4:
    sub      r6, r7
.clear4.1:
    cmp      4, r7
    bl       .clear4.2
    st.w     r0, 0[r6]
    add      4, r6
    add      -4, r7
    br       .clear4.1
.clear4.2:
    cmp      2, r7
    bl       .clear4.3
    st.h     r0, 0[r6]
    add      2, r6
    add      -2, r7
.clear4.3:
    cmp      0, r7
    bz       .clear4.4
    st.b     r0, 0[r6]
.clear4.4:
    jmp      [lp]
```

Processing up to this point is repeated for the number of sections that need to be initialized.

(4) Branch to main function

When the PIC facility is used and a jump is to be made to the main function with the FERET instruction, the ROM offset value is added to the value that is stored in FEPC.

```
    mov      #_exit, lp       ; lp <- #_exit
    mov      #_main, r10

    add      r29, lp          ; ROM offset value is added
    add      r29, r10         ; ROM offset value is added

    ldsr     r10, 2, 0        ; FEPC <- #_main

             ; apply PSW and PC to start user mode
    feret
```

Coding example

A coding example for using the PIC, PIROD, or PID facility is shown below.

```
$ifdef __PIC
  .TEXT .macro
    .section .pctext, pctext
  .endm
$else
  .TEXT .macro
    .section .text, text
  .endm
$endif
```

```
$ifdef __PID
  .STACK_BSS .macro
    .section .stack.bss, sbss32
  .endm
$else
  .STACK_BSS .macro
    .section .stack.bss, bss
  .endm
$endif

;-------------------------------------------------------------------------------
;   system stack
;-------------------------------------------------------------------------------
STACKSIZE    .set    0x200
    .STACK_BSS
    .align   4
    .ds      (STACKSIZE)
    .align   4
_stacktop:

;-------------------------------------------------------------------------------
;   startup
;-------------------------------------------------------------------------------
    .TEXT
    .public  __cstart
    .align   2
__cstart:

$ifdef __PIC
    jarl     .pic_base, r29
.pic_base:
    mov      #.pic_base, r10
    sub      r10, r29
$endif

$ifdef __PID
    mov      0xfedf0000, r28       ; Memory address for passing RAM offset value
    ld.w     0[r28], r28           ; Offset (RAM offset) between data arrangement
                                   ; at linkage and data arrangement at execution
$endif

    mov      #_stacktop, sp        ; set sp register
    mov      #__gp_data, gp        ; set gp register
    mov      #__ep_data, ep        ; set ep register
$ifdef __PID
    add      r28, sp
    add      r28, gp
    add      r28, ep
$endif

    ; initialize 1 data section
$ifdef __PID
  $ifdef __PIROD
    mov      #__s.sdata32, r6
    add      r29, r6
    mov      #__e.sdata32, r7
    add      r29, r7
    mov      #__s.sdata32.R, r8
    add      r28, r8
```

```
  $else
    mov      #__s.sdata32, r6
    mov      #__e.sdata32, r7
    mov      #__s.sdata32.R, r8
    add      r28, r8
  $endif
$else
  $ifdef __PIROD
    mov      #__s.data, r6
    add      r29, r6
    mov      #__e.data, r7
    add      r29, r7
    mov      #__s.data.R, r8
  $else
    mov      #__s.data, r6
    mov      #__e.data, r7
    mov      #__s.data.R, r8
  $endif
$endif
    jarl     _copy4, lp

    ; initialize 1 bss section
$ifdef __PID
    mov      #__s.sbss32, r6
    mov      #__e.sbss32, r7
    add      r28, r6
    add      r28, r7
$else
    mov      #__s.bss, r6
    mov      #__e.bss, r7
$endif
    jarl     _clear4, lp

    ; enable FPU
$if 1 ; disable this block when not using FPU
    stsr     6, r10, 1         ; r10 <- PID
    shl      21, r10
    shr      30, r10
    bz       .L1               ; detecting FPU
    stsr     5, r10, 0         ; r10 <- PSW
    movhi    0x0001, r0, r11
    or       r11, r10
    ldsr     r10, 5, 0         ; enable FPU

    movhi   0x0002, r0, r11
    ldsr    r11, 6, 0          ; initialize FPSR
    ldsr    r0, 7, 0           ; initialize FPEPC
.L1:
$endif

    ; set various flags to PSW via FEPSW

    stsr    5, r10, 0          ; r10 <- PSW
    ;xori   0x0020, r10, r10   ; enable interrupt
    ;movhi  0x4000, r0, r11
    ;or     r11, r10           ; supervisor mode -> user mode
    ldsr    r10, 3, 0          ; FEPSW <- r10
    mov     #_exit, lp         ; lp <- #_exit
    mov     #_main, r10
```

```
$ifdef __PIC
    add     r29, lp
    add     r29, r10
$endif
    ldsr    r10, 2, 0           ; FEPC <- #_main

    ; apply PSW and PC to start user mode
    feret

_exit:
    br      _exit               ;  end of program

;-------------------------------------------------------------------------------
;   copy routine
;-------------------------------------------------------------------------------
    ; r6: source begin  (4-byte aligned)
    ; r7: source end  (r6 <= r7)
    ; r8: destination begin  (4-byte aligned)
    .align  2
_copy4:
    sub     r6, r7
.copy4.1:
    cmp     4, r7
    bl      .copy4.2
    ld.w    0[r6], r10
    st.w    r10, 0[r8]
    add     4, r6
    add     4, r8
    add     -4, r7
    br      .copy4.1
.copy4.2:
    cmp     2, r7
    bl      .copy4.3
    ld.h    0[r6], r10
    st.h    r10, 0[r8]
    add     2, r6
    add     2, r8
    add     -2, r7
.copy4.3:
    cmp     0, r7
    bz      .copy4.4
    ld.b    0[r6], r10
    st.b    r10, 0[r8]
.copy4.4:
    jmp     [lp]

;-------------------------------------------------------------------------------
;   clear routine
;-------------------------------------------------------------------------------
    ; r6: destination begin (4-byte aligned)
    ; r7: destination end  (r6 <= r7)
    .align  2
_clear4:
    sub     r6, r7
.clear4.1:
    cmp     4, r7
    bl      .clear4.2
    st.w    r0, 0[r6]
    add     4, r6
    add     -4, r7
    br      .clear4.1
```

```
.clear4.2:
    cmp     2, r7
    bl      .clear4.3
    st.h    r0, 0[r6]
    add     2, r6
    add     -2, r7
.clear4.3:
    cmp     0, r7
    bz      .clear4.4
    st.b    r0, 0[r6]
.clear4.4:
    jmp     [lp]

;-------------------------------------------------------------------------------
;   dummy section
;-------------------------------------------------------------------------------
$ifdef __PID
    .section .sdata32, sdata32
.L.dummy.sdata32:
    .section .sbss32, sbss32
.L.dummy.sbss32:
$else
    .section .data, data
.L.dummy.data:
    .section .bss, bss
.L.dummy.bss:
$endif

$ifdef __PIROD
    .section .pcconst32, pcconst32
.L.dummy.pcconst32:
$else
    .section .const, const
.L.dummy.const:
$endif
;-------------------- end of start up module -------------------;
```

9. FUNCTION CALL INTERFACE SPECIFICATIONS

This chapter explains how to handle arguments when a program is called by the CC-RH.

9.1 Function Call Interface

This section describes how to handle arguments when a program is called by the CC-RH.

9.1.1 General-purpose registers guaranteed before and after function calls

Some general-purpose registers are guaranteed to be the same before and after a function call, and others are not. The rules for guaranteeing general-purpose registers are as follows.

(1) Registers guaranteed to be same before and after function call (Callee-Save registers)
These general-purpose registers must be saved and restored by the called function. It is thus guaranteed to the caller that the register contents will be the same before and after the function call.
r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30[Note], r31

Note r30 (EP) may be locked throughout the entire program. If it is locked, then the contents of the general-purpose registers are never changed anywhere in the program, and consequently it is not necessary for the callee to save and restore the registers.

(2) Registers not guaranteed to be same before and after function call (Caller-Save registers)
General-purpose registers other than the Callee-Save registers above could be overwritten by the called function. It is thus not guaranteed to the caller that the register contents will be the same before and after the function call.

Remark 1. The user must take responsibility for overwriting register r1, because it may be used by the assembler.

Remark 2. r2 may be reserved by the OS. The rules described here do not apply to reserved registers, because the compiler does not use them as general-purpose registers. The user is responsible for overwriting them.

Remark 3. r3 is a stack pointer. The rules described here do not apply to it, because it is not used as a general-purpose register. The user is responsible for overwriting it.

Remark 4. It is possible to specify usage of r2, r4, and r30 using options.

9.1.2 Setting and referencing arguments and return values

(1) Passing arguments
Arguments can be passed via registers or the stack. The manner in which each argument is passed is determined by the procedure below.

(a) A memory image to which each argument is assigned is created on the stack

<1> Scalar values that are 2 bytes or smaller are promoted to 4-byte integers before being stored.

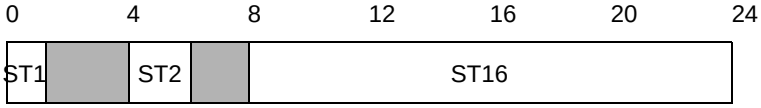
<2> Each argument is essentially aligned on a 4-byte boundary.

<3> If a return value is a structure or union, then the start of the memory image is set to the address at which to write the return-value data.

<4> If the function prototype is unknown, then each scalar-type argument is stored as follows.

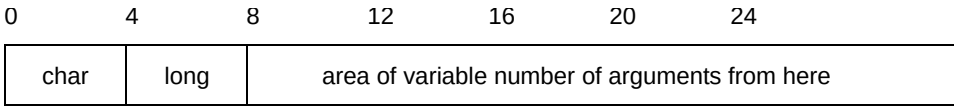
- 1-byte scalar integer ->Promoted to 4-byte integer, then stored
- 2-byte scalar integer ->Promoted to 4-byte integer, then stored
- 4-byte scalar integer ->Stored as-is
- 8-byte scalar integer ->Stored as-is
- 4-byte scalar floating-point number ->Promoted to 8-byte floating-point number, then stored
- 8-byte scalar floating-point number ->Stored as-is

Example 1. Function prototype : f(ST1, ST2, ST16)
Example where STx represents a structure with a size of x[byte]



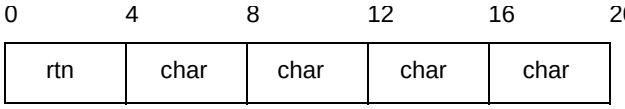
In the case of a structure or union whose size is not a multiple of 4, it is possible to add padding between the parameters. The contents of the padded area are undefined.

Example 2. Function prototype : f(char, long, ...)
Example of accepting variable number of actual arguments



The "area of variable number of arguments from here" consumes memory for the number of actual arguments that are set.

Example 3. Function prototype : ST4 f(char, char, char, char)



An address of the location to which to write the ST4 return value is passed through rtn.

(b) The first 4 words (16 bytes) of the created memory image are passed via registers r6 to r9, and the portion that does not fit is passed on the stack

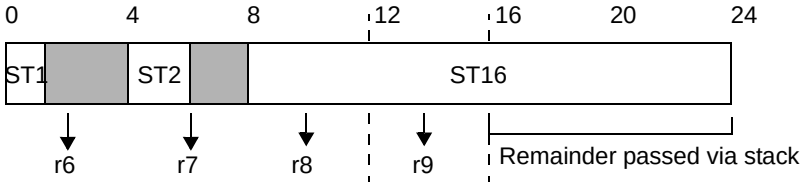
<1> If the arguments passed via the registers, it's loaded by the word units to each register (r6-r9).
The byte units and the half-word units aren't loaded.

<2> The arguments passed on the stack are set in the stack frame of the calling function.

<3> Arguments passed on the stack are stored on the stack in order from right to left in the memory image. Thus the word data at the 16-byte offset location of the memory image is placed in the location closest to 0.

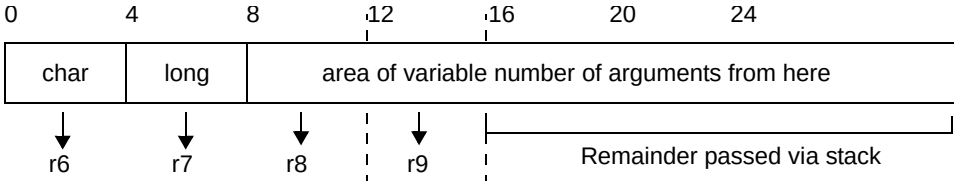
Remark See "9.1.4 Stack frame" about how data is placed on the stack.

Example 1. Function prototype : f(ST1, ST2, ST16)
Example where STx represents a structure with a size of x[byte]



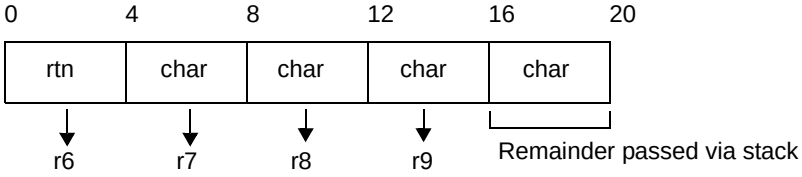
Even if only part of a structure (in this case, ST16) can fit in the registers, that part is still passed via the registers.

Example 2. Function prototype : f(char, long, ...)
Example of accepting variable number of actual arguments



Even if the number of arguments is variable, the arguments are passed via registers where this is possible.

Example 3. Function prototype : ST4 f(char, char, char, char)



Even if only passing four arguments of type char, the fourth argument may be passed on the stack, depending on the return value.

(2) How return values are passed
There are three ways to pass return values, as follows.

(a) If value is scalar type 4 bytes or smaller
The return value is returned to the caller via r10.
If the value is a scalar type less than 4 bytes in size, data promoted to 4 bytes is set in r10.
Zero promotion is performed on unsigned return values, and signed promotion on signed return values.

(b) If value is scalar type 8 bytes
The return value is returned to the caller via r10 and r11.
The lower 32 bits are set in r10, and the upper 32 bits in r11.

(c) If the value is a structure or union
If the return value is a structure or union, then when the caller calls the function, it sets the address to which to write the return value in the argument register r6. The caller sets the return value in the address location indicated by parameter register r6, and returns to the calling function.
Upon return, r6 and r10 are undefined (same as Caller-Save registers) to the calling function.
All structures and unions are turned by the same method, regardless of size. The actual data of the structure or union is not returned in the register.

9.1.3 Address indicating stack pointer

An address that is a multiple of 4 is set in the stack pointer.

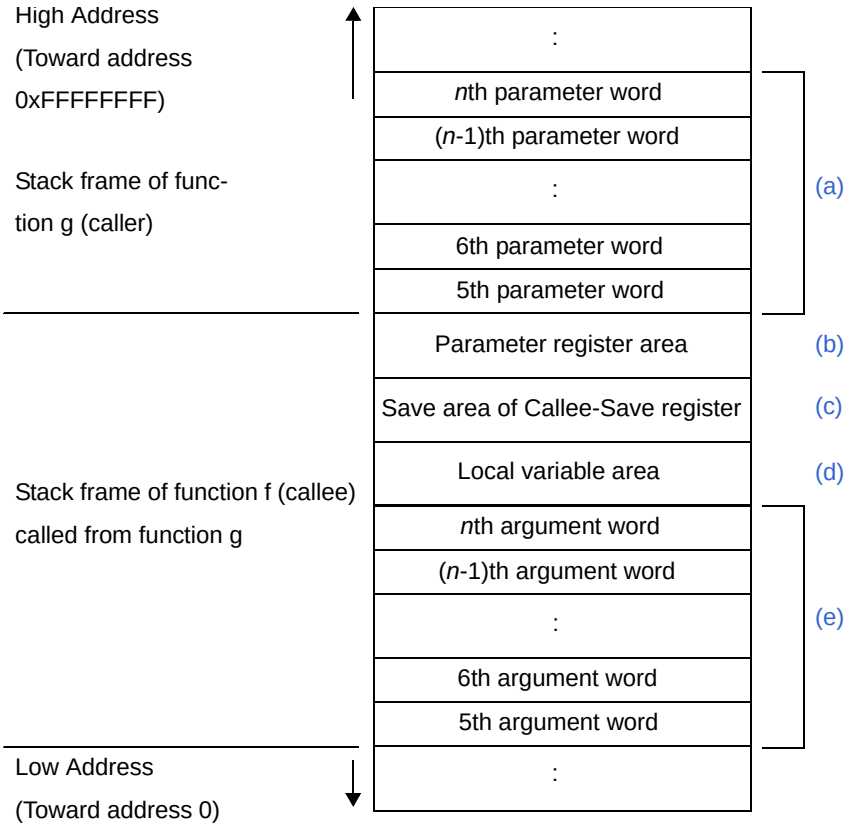
Although the addresses indicated by the stack pointer must all be either multiples of 4, it is not necessary for all the data stored on the stack to be aligned on either a 4-byte boundary. Each data item is stored on the stack at the location in accordance with its alignment. For example, if data is of type char, it can be stored on a 1-byte boundary even on the stack, because its data alignment is 1.

9.1.4 Stack frame

(1) Structure of the stack frame

Below is shown the stack frame of functions f and g from the perspective of function f, when function f is called by function g.

Figure 9.1 Contents of Stack Frame



Below is the range of the area that function f can reference and set.

(a) Parameter words 5 to *n*

This is the area where parameters beyond 4 words (16 bytes) are stored, when function f has parameters larger than 4 words in size. The size of this area is 0 when the parameter size is 4 words or less.

(b) Parameter register area

This area is for storing parameters passed in the registers (r6 to r9). The size is not locked at 16 bytes; the size is 0 if not needed.

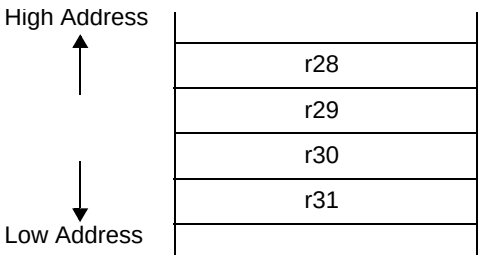
For details about the parameter register area, see "(2) Parameter register area".

(c) Save area of Callee-Save register

This area is for saving the Callee-Save registers used in function f. If it is necessary to save registers, then this area must be large enough for the number of registers.

Registers are essentially saved and restored using prepare/dispose instructions, so registered are stored in this save area in order of ascending register number.

For example, r28 to r31 would be saved in the following order.

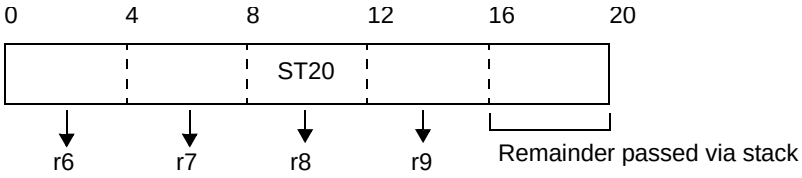


(d) Local variable area
This stack area is used for local variables.

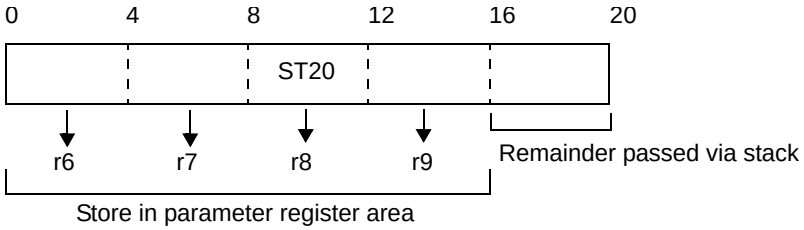
(e) 5th to *n*th argument words
Parameters beyond 4 words in size are stored in this area when function f is called by another function. The area for arguments needed when calling another function is allocated on function f's stack frame, and set there.
If fewer than 4 words are needed for the call's arguments, then the size of this area is 0.

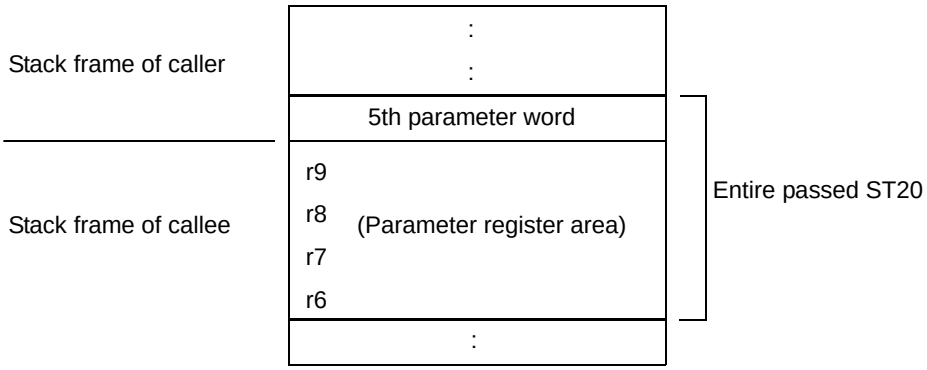
(2) Parameter register area
If the size of the parameters is greater than 4 words (16 bytes), then the required area for the size of the parameter register area is allocated. The size of this area will be either 0, 4, 8, 12, or 16 bytes, because it stores parameter registers r6 to r9, as necessary.
This area is for storing parameter registers when it is necessary to reference the contiguous placement relationship between the parameter register contents and parameters on the stack.
For example, when passing the value of a 20-byte structure argument, 16 bytes are passed in r6 to r9, and the remaining 4 bytes (the 5th parameter word) are passed on the stack.

Example Function prototype : f(ST20)



When referencing the value of the passed structure as a whole, it is necessary to align the entire structure contiguously in memory, but the structure is split unto the register portion and memory portion immediately after the function call.
In this case, the called function can reference the passed ST20 structure in memory be storing the parameter register on the stack.

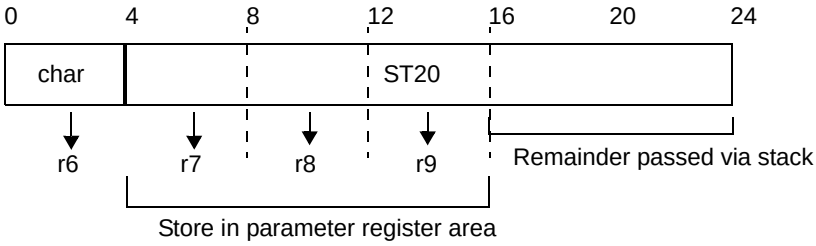




Below is a concrete case of parameters where this area is needed.
If none of these apply, then the parameter register area is not needed (size 0) because it is not necessary to store the parameter registers in the parameter register area.

(a) When a structure or union spans the parameter registers and stack

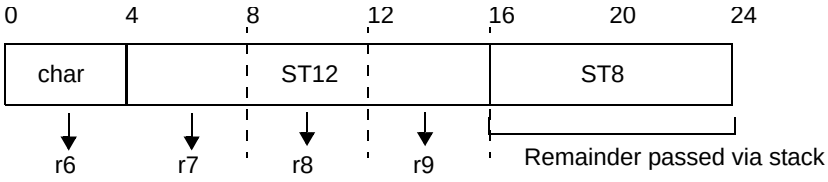
Example Function prototype : f(char, ST20)



In this case, r7 to r9 are stored in the parameter register area.
r6 is not stored because it is not needed to align ST20 contiguously in memory.
Therefore the size of the parameter register area is 12 bytes.

If a structure or union does not span the parameter register and stack, then it is not necessary to store it in the parameter register area, and the size of the parameter register area is therefore 0.

Example Function prototype : f(char, ST12, ST8)

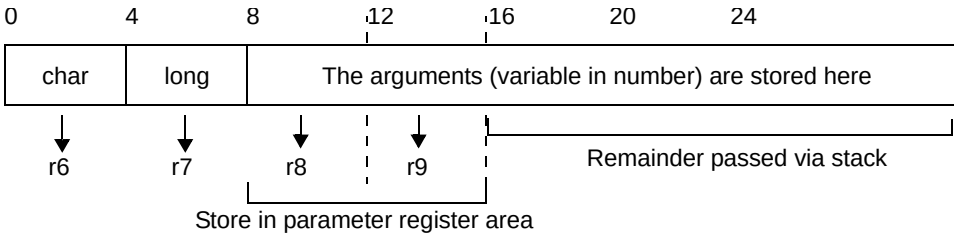


No store in parameter register area

In this case, all of ST12 fits in the parameter registers, ST8 is not passed in the parameter registers.
Since no arguments span the parameter registers and stack, the size of the parameter register area is 0 bytes.
If a structure or union is passed in its entirety via the parameter registers, the local variable area is used to expand it in memory.

(b) Accepting variable number of actual arguments
To receive a variable number of arguments, the arguments (including the last parameter) need to be stored in the parameter register area.

Example Function prototype : f(char, long, ...)



In this case, the parameter registers corresponding to the variable number of actual arguments (r8 and r9) are stored in the parameter register area.
Therefore the size of the parameter register area is 8 bytes.

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 by the CC-RH, they are prefixed with "_" (underscore) when they are output to the assembler.

Table 9.1 Identifier

C	Assembler
func1 ( )	_func1

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-RH 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 source 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, however, make sure that the value of SP is retained before and after calling.
When using a register variable register in an assembler function, make sure that the register value is retained before and after the assembler function is called. In other words, save the value of the register variable register before calling the assembler function, and restore the value after calling.
The register for register variable that can be used differ depending on the register mode.

Table 9.2 Registers for Register Variables

Register Modes	Register for Register Variable
22-register mode	r25, r26, r27, r28, r29
32-register mode	r20, r21, r22, r23, r24, r25, r26, r27, r28, r29

(3) Return address passed to C function

The CC-RH generates codes on the assumption that the return address of a function is stored in link pointer lp (r31). When execution branches to an assembler function, the return address of the function is stored in lp. Execute the jmp [lp] instruction to return to a C function.

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-RH 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) Work register

The CC-RH retains the values of the register for register variable before and after a C function is called but does not retain the values of the work registers. Therefore, do not leave a value that must be retained assigned to a work register.
The register for register variable and work registers that can be used differ depending on the register mode.

Table 9.3 Registers for Register Variables

Register Modes	Register for Register Variable
22-register mode	r25, r26, r27, r28, r29
32-register mode	r20, r21, r22, r23, r24, r25, r26, r27, r28, r29

Table 9.4 Work Register

Register Modes	Work Register
22-register mode	r10, r11, r12, r13, r14
32-register mode	r10, r11, r12, r13, r14, r15, r16, r17, r18, r19

(3) Return address returned to assembler function
The CC-RH generates codes on the assumption that the return address of a function is stored in link pointer lp (r31). When execution branches to a C function, the return address of the function must be stored in lp. Execution is generally branched to a C function using the jarl instruction.

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-RH assembler performs as follows.

```
        .public _i
        .public _c
        .dseg   DATA
_i:
        .db4    0x0
_c:
        .db     0x0
```

9.5 General-purpose Registers

How the CC-RH uses the general-purpose registers are as follows.

Table 9.5 Using General-purpose Registers

Register	Usage
r0	Used for operation as value of 0. Base register of .data/.bss section reference
r1	caller save register
r2	caller save register Reserved for system (OS) (Switched via option)
r3 (sp)	Stack pointer
r4 (gp)	Global pointer for PID Fixed
r5 (tp)	Global pointer for constant data caller save register
r6 to r19	caller save register
r20 to r29	callee save register
r30 (ep)	Element pointer Fixed or callee save register (Switched via option)
r31 (lp)	Link pointer callee save register

10. MESSAGE

This chapter describes message that CC-RH outputs.

10.1 General

This section describes internal error message, error message, fatal error message, information message and warning message that CC-RH 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 aren't included

```
message-type 05 message-number : message
```

Remark Following contents are output as the continued character string.
Message Types : 1 alphabetic character
Messages : 5 digits

10.3 Message Types

This section describes the message types displayed by CC-RH.
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 Messages

This section describes the messages displayed by CC-RH.

10.4.1 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 ccrh command is too large.
	[Action by User]	Divide the data input to the ccrh 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.

10.4.2 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.
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*".
E0511132	[Message]	Failed to create temporary folder.
E0511133	[Message]	The parameter for the "*option*" 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*.
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*)
E0511200	[Message]	Internal error(*error-information*).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.

E0512001	[Message]	Failed to delete a temporary file "*file*".
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.
E0520017	[Message]	Expected a "]".
E0520018	[Message]	Expected a ")".
E0520019	[Message]	Extra text after expected end of number.
E0520020	[Message]	Identifier "*character string*" is undefined.
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 ";".
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.

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.
E0520220	[Message]	Integral value does not fit in required floating-point type.
E0520221	[Message]	Floating-point value does not fit in required floating-point type.
E0520222	[Message]	Floating-point operation result is out of range.
E0520228	[Message]	Trailing comma is nonstandard.
	[Explanation]	A trailing comma is not standard.
E0520230	[Message]	Nonstandard type for a bit field.
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).
E0520268	[Message]	Declaration may not appear after executable statement in block.
E0520274	[Message]	Improperly terminated macro invocation.
E0520296	[Message]	Invalid use of non-lvalue array.
E0520301	[Message]	typedef name has already been declared (with same type).
E0520325	[Message]	inline specifier allowed on function declarations only.
E0520375	[Message]	Declaration requires a typedef name.
E0520393	[Message]	Pointer to incomplete class type is not allowed.
E0520404	[Message]	Function "main" may not be declared inline.
E0520409	[Message]	*Type* "*symbol*" returns incomplete type "*type*".
E0520411	[Message]	A parameter is not allowed
E0520450	[Message]	The type "long long" is nonstandard.
E0520469	[Message]	Tag kind of *character string1* is incompatible with declaration of *character string2*.
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.

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.
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.
E0520702	[Message]	Expected an "=".
E0520731	[Message]	Array with incomplete element type is nonstandard.
E0520749	[Message]	A type qualifier is not allowed.
E0520757	[Message]	*name* is not a type name.
E0520765	[Message]	Nonstandard character at start of object-like macro definition.
E0520816	[Message]	In a function definition a type qualifier on a "void" return type is not allowed.
E0520852	[Message]	Expression must be a pointer to a complete object type.
E0520861	[Message]	Invalid character in input line.
E0520862	[Message]	Function returns incomplete type "*type*".
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".
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.
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 identifier.
E0521038	[Message]	Expected a string literal.
E0521039	[Message]	Unrecognized STDC pragma.
E0521040	[Message]	Expected "ON", "OFF", or "DEFAULT".
E0521045	[Message]	Invalid designator kind.
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*.
E0521072	[Message]	A declaration cannot have a label.
E0521144	[Message]	Storage class must be auto or register.
E0521158	[Message]	Void return type cannot be qualified.
E0521260	[Message]	Invalid alignment specifier value.
E0521261	[Message]	Expected an integer literal.
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.
E0521578	[Message]	case label value has already appeared in this switch at line *number*.
E0521584	[Message]	Parentheses around a string initializer are nonstandard.
E0521649	[Message]	White space is required between the macro name *name* and its replacement text.
E0523005	[Message]	Invalid pragma declaration
	[Explanation]	Write the #pragma syntax in accord with the correct format.
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.
E0523026	[Message]	Incorrect PIC address usage.
E0523027	[Message]	Incorrect PID address usage.
E0523048	[Message]	Illegal reference to interrupt function.
E0523057	[Message]	Illegal section specified
	[Explanation]	Strings that are not usable for the purpose were used to specify the attributes of sections.
E0523058	[Message]	Illegal #pragma section syntax
	[Explanation]	The #pragma section syntax is illegal.

E0523065	[Message]	Cannot assign address constant to initializer for bitfield.
E0523066	[Message]	The combination of the option and section specification is inaccurate
E0523067	[Message]	Type nest is too deep.
E0523069	[Message]	Two or more "pm numbers" cannot be used
E0523070	[Message]	The "cmn" designated variable can be accessed only by r0 relativity
E0523071	[Message]	The "cmn" specification function can access the static variable only with r0 relativity
E0523072	[Message]	The "cmn" specification function can call the "pmodule" specified function only with "cmn" specification
E0523073	[Message]	*name* does not support this intrinsic function.
E0523087	[Message]	Illegal reference to "*function name*"
E0523090	[Message]	A parameter may not have __fp16 type.
E0523091	[Message]	Function returning __fp16 is not allowed.
E0523118	[Message]	Element name is illegal or missing.
E0523119	[Message]	Taking the address of a vector element is not allowed.
E0523122	[Message]	Incorrect PIROD address usage.
E0523123	[Message]	A function cannot be defined in this section.
E0523124	[Message]	A variable cannot be defined in this section.
E0523125	[Message]	A string literal cannot be used in this section.
E0523126	[Message]	"cmn" cannot be specified for a function in this section.
E0523127	[Message]	"cmn" cannot be specified for a string literal in this section.
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.
E0550207	[Message]	Illegal expression (labels have different reference types).
	[Action by User]	Check the expression.
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.
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.
E0550239	[Message]	Illegal operand (cannot use r0 as source in RH850 mode).
	[Action by User]	Check the operand.
E0550240	[Message]	Illegal operand (cannot use r0 as destination in RH850 mode).
	[Action by User]	Check the operand.

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.
E0550261	[Message]	Illegal condition code.
	[Explanation]	Illegal condition code. 0xd cannot be specified for the condition code of an adf.sbf instruction.
	[Action by User]	Check the condition code.
E0550265	[Message]	Illegal register number (r0-r7, r16-r31).
	[Action by User]	You can only specify one of r8 to r15 as the general-purpose register. Check the operand.
E0550267	[Message]	Illegal operand (displacement must be multiple of 4).
	[Action by User]	Check the displacement.
E0550269	[Message]	Illegal mnemonic(cannot use this mnemonic ins RH850 "*core-name*" core).
	[Explanation]	The mnemonic is not supported by the selected core.
	[Action by User]	Check the mnemonic.
E0550270	[Message]	The same register must not be used for the first and second operands.
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.
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.
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.
E0550652	[Message]	The "*option1*" option needs the "*option2*" option.
	[Explanation]	In order to use "*option 1*", "*option 2*" needs to be specified simultaneously.
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.
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.
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.
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.
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 now allowed.
E0551311	[Message]	Specified value is out of range 1-7.
E0551313	[Message]	"*reg*" is not allowed.
	[Explanation]	Register *reg* is not allowed here.
	[Action by User]	Check which operands are allowed.
E0551401	[Message]	Illegal operand "*string*".
E0551402	[Message]	Illegal instruction.
E0551403	[Message]	Illegal operand of .DB8 directive.
E0551406	[Message]	Any symbol name starting with a period must not be used for "*string*".
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.
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.
E0562200	[Message]	Illegal object file : "*file*"
	[Explanation]	A format other than ELF format was input.
E0562201	[Message]	Illegal library file : "*file*"
	[Explanation]	*file* is not a library 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 link-time(inter-module) optimization information.
	[Action by User]	Do not specify the goptimize option at compilation or assembly.
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.
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. 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.
E0562324	[Message]	Section "*section*" in "*file*" conflicts
	[Explanation]	More than one object file containing "*section*" was input.
E0562326	[Message]	Insufficient space to allocate prefetch section after section "*section*"
	[Explanation]	Insufficient space to allocate prefetch section after section "*section*"
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.
E0562408	[Message]	Register mode in "*file*" conflicts with that in another file("*mode*")
	[Explanation]	Different register modes are specified across multiple files.
	[Action by User]	Check the options used on compiling.
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.

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*.
E0562430	[Message]	Register ("*register1*") in "*file-name*" conflicts with that in another file("*register2*")
	[Explanation]	The usage method of the register mode that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562431	[Message]	GP register ("mode = *mode1*") in "*file-name*" conflicts with that in another file("mode = *mode2*")
	[Explanation]	The usage method of the GP register that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562432	[Message]	EP register ("mode = *mode1*") in "*file-name*" conflicts with that in another file("mode = *mode2*")
	[Explanation]	The usage method of the EP register that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.

E0562433	[Message]	TP register ("mode = *mode1*") in "*file-name*" conflicts with that in another file("mode = *mode2*")
	[Explanation]	The usage method of the TP register that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562434	[Message]	R2 register ("mode = *mode1*") in "*file-name*" conflicts with that in another file("mode = *mode2*")
	[Explanation]	The usage method of the R2 register that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562435	[Message]	Alignment of 8byte data (value="*alignment1*") in "*file-name*" conflicts with that in another file(value="*alignment2*")
	[Explanation]	The alignment condition for the 8-byte basic type in C language that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562436	[Message]	Size of double/long double (value="*size1*") in "*file-name*" conflicts with that in another file(value="*size2*")
	[Explanation]	The size for the 8-byte basic type in C language that was specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562437	[Message]	FPU Type (value="*FPU1*") in "*file-name*" conflicts with that in another file(value="*FPU2*")
	[Explanation]	The FPU type specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562438	[Message]	SIMD Type (value="*SIMD1*") in "*file-name*" conflicts with that in another file(value="*SIMD2*")
	[Explanation]	The SIMD type specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562439	[Message]	Number of additional global pointers (value="*number1*") in "*file-name*" conflicts with that in another file(value="*number2*")
	[Explanation]	The number of GP registers specified in "*file-name*" does not match that in other files.
	[Action by User]	Check the options used on compiling.
E0562450	[Message]	Illegal ID(value="*number*") in section "*section-name*" in "*file-name*"
	[Explanation]	"*number*" specified by "*section-name*" in "*file-name*" is not supported.
	[Action by User]	Check if the compiler and assembler versions are correct.
E0562451	[Message]	Illegal desc(number) in section "*section-name*" in "*file-name*"
	[Explanation]	"*number*" specified by "*section-name*" in "file-name" is not supported.
	[Action by User]	Check if the compiler and assembler versions are correct.
E0562600	[Message]	Library "*library*" requires "*edition* EDITION"
	[Explanation]	The "*library*" requires the "*edition*" edition.

E0595001	[Message]	checker : Missing input file
	[Explanation]	No file name is specified.
E0595002	[Message]	checker : Failed to open input file "*file name*"
	[Explanation]	The file cannot be opened.
E0595003	[Message]	checker : Incorrect usage
	[Explanation]	There is an error in the specification on the command line.
E0595004	[Message]	checker : Incorrect file format
	[Explanation]	A file that is not a Motorola S-type file is specified.
E0595005	[Message]	checker : Failed to decode input file "*file name*"
	[Explanation]	Decoding failed during parsing. The file format may be incorrect or the setting of the base address for an exception vector may be incorrect.
	[Action by User]	See "11.4 Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler".
E0595010	[Message]	NG : *name* : *address* : *cause*
	[Explanation]	No SYNCP instruction may be allocated between the location of the exception handler for the exception source *name* and the location *address*.
	[Action by User]	See "11.4 Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler".

10.4.3 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 *file name*.
F0520164	[Message]	Name of directory for temporary files is too long *file name*.
F0520182	[Message]	Could not open source file *file name* (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: *file name*.
F0523029	[Message]	Cannot open rule file
	[Explanation]	The file specified in the -Xmisra2004="*file name*" or -Xmisra2012="*file name*" option cannot be opened.
F0523030	[Message]	Incorrect description "*file name*" in rule file
	[Explanation]	The file specified in the -Xmisra2004="*file name*" or -Xmisra2012="*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.
F0523054	[Message]	regID is out of range
	[Action by User]	Specify an usable value as regID.
F0523055	[Message]	selID is out of range
	[Action by User]	Specify an usable value as selID.
F0523056	[Message]	NUM is out of range
	[Explanation]	A value that is not usable as NUM in __set_il_rh(NUM, ADDR) was specified.
F0523061	[Message]	argument is incompatible with formal parameter of intrinsic function
F0523062	[Message]	return value type does not match the intrinsic function

F0523073	[Message]	*core name* does not support this intrinsic function.
	[Explanation]	An intrinsic function not usable in the specified core was used.
F0523089	[Message]	Cannot read file "*file name*".
F0530320	[Message]	Duplicate symbol "*symbol name*".
F0530321	[Message]	Section "*section name*" conflicts.
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.
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.
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 2 Gbytes.
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 #pragma "*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.
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.
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*"

F0563004	[Message]	Invalid inter-module optimization information type in "*file*"
	[Explanation]	The file contains an unsupported link-time(inter-module) optimization information *type*.
	[Action by User]	Check if the compiler and assembler versions are 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*" [V1.05.00 or earlier] Section contents overlap in absolute section "*section*" in "*file*" [V1.06.00 or later]
	[Explanation]	Data addresses overlap within an absolute address section.
	[Action by User]	Modify the source program.
F0563103	[Message]	Section size overflow : "*section-name*"
	[Explanation]	Section "*section-name*" 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.
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.
F0563150	[Message]	Multiple files cannot be specified while processing "*process*"
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.
F0563431	[Message]	Incorrect device type, object file mismatch.
F0563600	[Message]	Option "*option*" requires parameter
F0563601	[Message]	Invalid parameter specified in option "*option*" : "*parameter*"
F0563602	[Message]	"*character string*" option requires "*edition*".
	[Explanation]	The "*character string*" option requires the "*edition*".

10.4.4 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.
M0536001	[Message]	control register is written. (*register-information*)
	[Explanation]	Writing to a control register has been 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 link-time(inter-module) optimization information was found in *file*. link-time(Inter-module) optimization is not performed on *file*. To perform link-time(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*.
M0560512	[Message]	Section "*section*" created by "*option*"
M0560700	[Message]	Section address overflow out of range : "*section*"
	[Explanation]	The address of "*section*" is beyond the allowable address range.

10.4.5 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.
W0511143	[Message]	The "-Xfloat" option is ignored because specified device does not have FPU.
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*".
	[Explanation]	The same file name was specified more than once in a command line. CC-RH is not capable of handling multiple instances of the same file name. Only the last file name to have been specified is valid.
W0511179	[Message]	The evaluation version is valid for the remaining *number* days.
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 "*option*" 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.
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.
W0520147	[Message]	Declaration is incompatible with "*declaration*" (declared at line *number*).
W0520152	[Message]	Conversion of nonzero integer to pointer.
W0520159	[Message]	Declaration is incompatible with previous *name*.
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.
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.
W0520220	[Message]	Integral value does not fit in required floating-point type.
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 *name* 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 variable "*name*" 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 *name1* cannot be assigned to an entity of type *name2*.
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.
W0520609	[Message]	This kind of pragma may not be used here.
W0520618	[Message]	struct or union declares no named members.
W0520660	[Message]	Invalid packing alignment value.
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 *name*.
W0520870	[Message]	Invalid multibyte character sequence.
W0520940	[Message]	Missing return statement at end of non-void function "*name*".
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.
W0521105	[Message]	#warning directive: *character string*.
	[Explanation]	There is a "#warning" directive in the source file.
W0521222	[Message]	Invalid error number.
W0521223	[Message]	Invalid error tag.
W0521224	[Message]	Expected an error number or error tag.
W0521273	[Message]	Alignment-of operator applied to incomplete type.
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.
W0523038	[Message]	A struct/union/class has different pack specifications.
W0523042	[Message]	Using "*function item*" function at influence the code generation of "SuperH" compiler
	[Action by User]	The use of "*function item*" may affect compatibility with the SuperH compiler. Confirm details of differences in the specification.

W0523061	[Message]	argument is incompatible with formal parameter of intrinsic function
W0523062	[Message]	return value type does not match the intrinsic function
W0523068	[Message]	Atomic transfer function is used in "*cpu*"
W0523116	[Message]	"*character string*" and other settings are inconsistent.
W0530809	[Message]	const qualifier for variable "*variable-name*" differs between files.
W0530811	[Message]	Type of symbol "*symbol-name*" differs between files.
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.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.
W0550014	[Message]	Illegal list value, ignored.
	[Explanation]	Illegal list value, ignored. Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the register list value.
W0550015	[Message]	Illegal register number, ignored.
	[Explanation]	Illegal register number, ignored. The invalid register will be ignored, and the assembly will continue.
	[Action by User]	Check the register list register.
W0550017	[Message]	Base register is ep(r30) only.
	[Action by User]	Check the base register specification.
W0550018	[Message]	Illegal regID for *inst*.
	[Action by User]	Check the system register number.

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.
W0550021	[Message]	*string* already specified, ignored.
	[Explanation]	*string* already specified, ignored. The previously specified number will be used. This specification will be ignored.
	[Action by User]	Check the number of registers.
W0550026	[Message]	Illegal register number, aligned odd register(r*XX*) to be even register(r*YY*).
	[Explanation]	Odd-numbered registers (r1, r3, ... r31) have been specified. The only general-purpose registered that can be specified are even-numbered (r0, r2, r4, ... r30). Assembly will continue, assuming that even-numbered registers (r0, r2, r4, ... r30) have been specified.
	[Action by User]	Check the register specification.
W0550028	[Message]	Duplicated reg_mode, ignored $REG_MODE.
	[Explanation]	Duplicated reg_mode, ignored $REG_MODE. The "-Xreg_mode" option takes precedence, and register modes specified via the $REG_MODE control instruction will be ignored.
	[Action by User]	Check the option specification.
W0550031	[Message]	*identifier* undefined.
	[Action by User]	Check the identifier.
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 messages 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.
W0561016	[Message]	The evaluation version of *version* is valid for the remaining *number* days. After that, link size limit (256 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.
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 the *option* 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: The "-output" option does not accept specification of a section that has no initial value.

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.
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*"
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"
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.
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*.

W0561210	[Message]	Section "*section-name*" ID(value="*number*") in "*file-name*" is reserved
	[Explanation]	There are numbers that were assigned to be reserved in the section information in the input file. The "*number*" specification will be ignored. Check if the compiler and assembler versions are correct.
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 link-time(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 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.
W0561326	[Message]	Reserved symbol "*symbol*"is defined in "*file*"
	[Explanation]	Reserved symbol name *symbol* is defined in *file*.
W0561327	[Message]	Section alignment in option "aligned_section" is small : "*section*"
	[Explanation]	Since the boundary alignment value specified for aligned_section is 16 which is smaller than that of *section*, the option settings made for that section are ignored.

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.
W0561510	[Message]	Input file was compiled with option "smap" and option "map" is specified at linkage
	[Explanation]	A file was compiled with smap specification.
	[Action by User]	The file with smap specification should not be compiled with the map option specification in the second build processing.
W0595020	[Message]	Warning : *name* : *address* : *cause*
	[Explanation]	No SYNCP instruction may be allocated at the location of the exception handler for the exception source *name*.
	[Action by User]	See "11.4 Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler".

11. CAUTIONS

This chapter explains the points to be noted when using the CC-RH.

11.1 Volatile Qualifier

When a variable is declared with the volatile qualifier, the variable is not optimized and optimization for assigning the variable to a register is no longer performed. When a variable with volatile specified is manipulated, a code that always reads the value of the variable from memory and writes the value to memory after the variable is manipulated is output. The access width of the variable with volatile specified is not changed.

A variable for which volatile is not specified is 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 to variables for which volatile is not specified, the instruction may be deleted as a result of optimization because it is interpreted as a redundant instruction. The volatile qualifier must be specified especially for variables that access a peripheral I/O register, variables whose value is changed by interrupt servicing, or variables whose value is changed by an external source.

The following problem may occur if volatile is not specified where it should.

- The correct calculation result cannot be obtained.
- Execution cannot exit from a loop if the variable is used in a for loop.
- The order in which instructions are executed differs from the intended order.
- The number times memory is accessed and the width of access are not as intended.

If it is clear that the value of a variable with volatile specified is not changed from outside in a specific section, the code can be optimized by assigning the unchanged value to a variable for which volatile not specified and referencing it, which may increase the execution speed.

Example Source and output code if volatile is not specified

If volatile is not specified for "variable a", "variable b", and "variable c", these variables are assigned to registers and optimized. For example, even if an interrupt occurs in the meantime and the variable value is changed by the interrupt, the changed value is not reflected.

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        MOVHI   HIGHW1(#_a), R0, R6
        LD.W    LOWW(#_a)[R6], R6
        CMP     0x00000000, R6
        MOVHI   HIGHW1(#_b), R0, R6
        LD.W    LOWW(#_b)[R6], R6
        BGT     .BB1_2          ; bb3
.BB1_1:     ; bb1
        ADD     0x00000001, R6
        BR      .BB1_3          ; bb9
.BB1_2:     ; bb3
        ADD     0x00000002, R6
.BB1_3:     ; bb9
        ADD     0x00000001, R6
        MOVHI   HIGHW1(#_b), R0, R7
        ST.W    R6, LOWW(#_b)[R7]
        JMP     [R31]
```

Example Source and output code if volatile is specified

If volatile is specified for "variable a", "variable b", and "variable c", a code that always reads the values of these variables from memory and writes them to memory after the variables are manipulated is output. For example, even if, an interrupt occurs in the meantime and the values of the variables are changed by the interrupt, the result in which the change is reflected can be obtained. (In this case, interrupts may have to be disabled while the variables are manipulated, depending on the timing of the interrupt.)

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:
        MOVHI   HIGHW1(#_a), R0, R6
        LD.W    LOWW(#_a)[R6], R6
        CMP     0x00000000, R6
        BGT     .BB1_2          ; bb3
.BB1_1:    ; bb1
        MOVHI   HIGHW1(#_b), R0, R6
        LD.W    LOWW(#_b)[R6], R6
        ADD     0x00000001, R6
        BR      .BB1_3          ; bb9
.BB1_2:    ; bb3
        MOVHI   HIGHW1(#_b), R0, R6
        LD.W    LOWW(#_b)[R6], R6
        ADD     0x00000002, R6
.BB1_3:    ; bb9
        MOVHI   HIGHW1(#_b), R0, R7
        ST.W    R6, LOWW(#_b)[R7]
        LD.W    LOWW(#_b)[R7], R6
        ADD     0x00000001, R6
        ST.W    R6, LOWW(#_b)[R7]
        JMP     [R31]
```

11.2 -Xcpu Option Specification for Assembler

The instructions that can be assembled differ depending on the parameter specified for the -Xcpu option. See the user's manual of the target device for the usable instructions. When an unusable instruction is written, an assemble error will occur and the following message will be output.

E0550269 : Illegal mnemonic(cannot use this mnemonic ins RH850 "*core-name*").

11.3 Controlling the Output of Bit Manipulation Instructions [V1.05.00 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 a 1-byte type.

(c) The bit field where the value is assigned is qualified with volatile.

To stop the output of bit manipulation instructions, satisfy condition (c) above and either assign a value that is not a constant in condition (a) or use a type that is not a 1-byte type in condition (b).

When none of the above conditions is satisfied, the compiler automatically determines whether to output bit manipulation instructions according to the specified optimization level and the contents of the source program.

Note 1-byte types are char, unsigned char, signed char, and _Bool.

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. */
}
```

11.4 Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler

The syncp_checker is a tool for confirming that the SYNCP instruction is inserted at the beginning of the exception handler when a Motorola S-type file is output by the build process for the G3M core project. This tool is available when CS+ V4.00.00 or a later version is installed.

For insertion of the SYNCP instruction at the beginning of the exception handler, see "RH850G3M User's Manual: Software" (Rev.1.10 or later).

This tool starts checking the Motorola S-type file from the base address (default address is 0) of the specified exception vector and returns one of the following results.

OK	A SYNCP instruction is inserted or no SYNCP instruction is necessary.
E0595010:NG	A SYNCP instruction is necessary but might not be inserted.
W0595020:Warning	This tool cannot determine whether the SYNCP instruction insertion is correct.

When NG or Warning is returned, check the code at the beginning of the exception handler.

Output files will be generated even if NG or Warning is returned.

In the CS+ integrated development environment, this tool can be controlled in the [Hex Output Options] tabbed page on the Property panel of the CC-RH build tool. For the [Confirm that SYNCP is inserted at entry of exception handler] property under the [Others] category, select [No] to stop activation of this tool. When the project does not include the exception vector table, use this setting to stop the tool.

This tool has the following options.

-b=*address*	Specify the base address of the exception vector as the start location of the check. The default value is 0.
-n=*num*	Specify the number of entries in the exception vector table that correspond to the user interrupt (EIINT). The default value is 16.

11.5 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, 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 expansion functions for more efficient use of the CC-RH.

A.1 Variables (C Language)

This section explains variables (C language).

A.1.1 Allocating to sections accessible with short instructions

CC-RH normally uses two instructions (for a total of 8 bytes) to access variables: a movhi instruction (4 bytes) and an ld/st instruction (4 bytes). By using a #pragma section directive, however, it generates code to access variables using one instruction: an ld/st instruction (4 or 6 bytes) or a sld/ss
t instruction (2 bytes). This makes it possible to reduce the code size. See below for details.

A.1.1.1 GP relative access

This generates code to access variables using one instruction by placing variables in sections that can be accessed via the global pointer (GP) and an ld/st instruction.

Use a #pragma section directive when defining or accessing variables, and specify either gp_disp16 or gp_disp23 as the attribute strings.

Note gp_disp32 does not have an effect to reduce the code size.

```
#pragma section attribute-strings
variable-declaration/definition
#pragma section default
```

Example 1. Accessing via a GP-relative 4-byte load/store instruction

```
#pragma section gp_disp16
int a = 1;              /*allocated to .sdata section*/
int b;                  /*allocated to .sbss section*/
#pragma section default
```

Example 2. Accessing via a GP-relative 6-byte load/store instruction

```
#pragma section gp_disp23
int a = 1;             /*allocated to .sdata23 section*/
int b;                 /*allocated to .sbss23 section*/
#pragma section default
```

A.1.1.2 EP relative access

You can reduce the code size by locating variables in a section that can be accessed via the element pointer (EP) and a sld/sst instruction or ld/st instruction. You can locate variables in a section that can be accessed relative to the EP using the following methods.

(1) Specifying the -Omap/-Osmap option
This optimizes access to external variables. It outputs code that accesses frequently accessed external variables relative to the EP.

(2) #pragma section directive
Use a #pragma section directive when defining or accessing variables, and specify either ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23, or ep_auto as the attribute string.

Note ep_disp32 does not have an effect to reduce the code size.

```
#pragma section attribute-strings
variable-declaration/definition
#pragma section default
```

Example 1. Accessing via a EP-relative 2-byte load/store instruction

```
#pragma section ep_disp4
int a = 1;              /*allocated to .tdata4 section*/
int b;                  /*allocated to .tbss4 section*/
#pragma section default
```

Even if ep_disp5, ep_disp7, or ep_disp8 is specified as the attribute string, access is via an EP-relative 2-byte load/store instruction (i.e. is the same as in the case of ep_disp4).

Example 2. Accessing via a EP-relative 4-byte load/store instruction

```
#pragma section ep_disp16
int a = 1;              /*allocated to .edata section*/
int b;                  /*allocated to .ebss section*/
#pragma section default
```

Example 3. Accessing via a EP-relative 6-byte load/store instruction

```
#pragma section ep_disp23
int a = 1;              /*allocated to .edata23 section*/
int b;                  /*allocated to .ebss23 section*/
#pragma section default
```

A.1.2 Changing allocated section

The default allocation sections of variables are as follows:

- Variables with no initial value: .bss section
- Variables with initial value: .data section
- const constants: .const section

A.1.2.1 Changing the area to be allocated using the #pragma section directive

To change the allocated section, specify the attribute strings using #pragma section directive.

Example #pragma section directive description

```
#pragma section gp_disp16 "mysdata"
int a = 1;              /*allocated to mysdata.sdata attribute section*/
int b;                  /*allocated to mysdata.sbss attribute section*/
#pragma section default
```

See "4.2.6.1 Allocation of function and data to section" for details about how to use the #pragma section directive.

When referencing a variable using the #pragma section directive from a function in another source file, it is necessary to declare the affected variable with the extern specifier and the same #pragma section directive in the file that performs reference.

Unlike when specifying a variable by means of a definition or declaration, it outputs the following error if the variable cannot be accessed with the specified section attribute.

E0562330 : Relocation size overflow : "*file*"-"*section*"-"*offset*"

Example 1. File that defines a table

```
#pragma section zconst
const unsigned char table_data[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9}; /*allocated to .zconst
section*/
#pragma section default
```

Example 2. File that references a table

```
#pragma section zconst
extern  const unsigned char  table_data[];      /*allocated to .zconst section*/
#pragma section default
```

Code such as the following can be used if portability of C source to the SH family of compilers is a concern.

Example #pragma section directive description

```
#pragma section mydata
int a = 1;              /*allocated to mydata.data section*/
int b;                  /*allocated to mydata.bss section*/
#pragma section default
```

A.1.2.2 Changing the area to be allocated using the -Xsection option

The -Xsection option can be used to change the default allocation section. The code size can be reduced by performing allocation to a section with high access efficiency.

(1) Specify the default section type using the -Xsection option.

Example Allocating to .sdata/.sbss section

```
>ccrh main.c -Xsection=data=gp_disp16
```

However, an error will be output at linkage if the variables do not fit in the section specified in (1). In this case, change the section of the variables in the C source file.

```
E0562330 : Relocation size overflow : "file"-"section"-"offset"
```

Example Changing variables to .sdata23/.sbss23 section

```
int a = 1;              /* Allocated to the section specified in (1) */
int b;                  /* Allocated to the section specified in (1) */
#pragma section gp_disp23
int c = 1;              /* Allocated to the .sdata23 section */
int d;                  /* Allocated to the .sbss23 section */
#pragma section default
int e = 1;              /* Allocated to the section specified in (1) */
int f;                  /* Allocated to the section specified in (1) */
```

A.1.2.3 Change the allocated area using the -Xpreinclude option

You can use the -Xpreinclude option to allocate all variables declared or defined in a file into an arbitrary section, without changing the C source file. You can reduce the code size by allocating them in a section with efficient access.

(1) Prepare a header file (.h) containing a #pragma section directive.

Example Allocating in .sdata/.sbss section [section.h]

```
#pragma section gp_disp16
```

(2) Use the -Xpreinclude option to include the header you created in (1) at the beginning of the compilation unit.

Example If the header file with the specified section is section.h

```
>ccrh main.c -Xpreinclude=section.h
```

Compiled as if main.c starts with an include of "section.h".

However, a link-time error will be output if the variables do not fit in the section specified in (1). In this case, change the section of the variables in the C source file.

```
E0562330 : Relocation size overflow : "file"-"section"-"offset"
```

Example Changing variables to .sdata23/.sbss23 section

```
int a = 1;              /*Allocated in section specified in (1)*/
int b;                  /*Allocated in section specified in (1)*/
#pragma section gp_disp23
int c = 1;              /*Allocated in .sdata23 section*/
int d;                  /*Allocated in .sbss23 section*/
#pragma section default
int e = 1;              /*Allocated in default .data section*/
int f;                  /*Allocated in default .bss section*/
```

A.1.3 Defining variables for use during standard and interrupt processing

Specify as volatile variables that are to be used during both standard and interrupt processing.

When a variable is defined with the volatile qualifier, the variable is not optimized. When manipulating variables specified as volatile, always read the value from memory, and when substituting the value, always write the value to memory. You cannot change the access order or access width of variables specified as volatile. A variable for which volatile is not specified is 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 to variables for which volatile is not specified, the 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 are assigned to a register, and may be optimized. If, for example, an interrupt occurs within this code, and a variable value is modified within the interrupt, the value will not be reflected.

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movhi   highw1(#_a), r0, r6
        ld.w    loww(#_a)[r6], r6
        cmp     0x00000000, r6
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        bgt     .bb1_2          ; bb3
.bb1_1:     ; bb1
        add     0x00000001, r6
        br      .bb1_3          ; bb9
.bb1_2:     ; bb3
        add     0x00000002, r6
.bb1_3:     ; bb9
        add     0x00000001, r6
        movhi   highw1(#_b), r0, r7
        st.w    r6, loww(#_b)[r7]
        jmp     [r31]
```

Example 2. Source and output code when volatile has been specified
If the volatile qualifier is specified for variables a, b, and c, the output code is such that the values of these variables are read from and written to memory whenever they must be assigned new values. Even if an interrupt occurs in the meantime and the values of the variables are changed by the interrupt, for example, the result in which the change is reflected can be obtained.
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:
        movhi   highw1(#_a), r0, r6
        ld.w    loww(#_a)[r6], r6
        cmp     0x00000000, r6
        bgt     .bb1_2          ; bb3
.bb1_1:    ; bb1
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        add     0x00000001, r6
        br      .bb1_3          ; bb9
.bb1_2:    ; bb3
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        add     0x00000002, r6
.bb1_3:    ; bb9
        movhi   highw1(#_b), r0, r7
        st.w    r6, loww(#_b)[r7]
        ld.w    loww(#_b)[r7], r6
        add     0x00000001, r6
        st.w    r6, loww(#_b)[r7]
        jmp     [r31]
```

A.1.4 Defining const constant pointer

The pointer is interpreted differently depending on the "const" specified location.
To assign the const section to the zconst section, specify #pragma section zconst. To assign the const section to the zconst23 section, specify #pragma section zconst23.

- const char *p;
  This indicates that the object (*p) indicated by the pointer cannot be rewritten.
  The pointer itself (p) can be rewritten.
  Therefore the state becomes as follows 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) indicated by the pointer can be rewritten.
  Therefore the state becomes as follows and the pointer itself is allocated to ROM (.const/.zconst/.zconst23).

```
*p = 0;     /*correct*/
p = 0;      /*error*/
```

- const char *const p;
  This indicates that neither the pointer itself(p) nor the object (*p) indicated by the pointer can be rewritten.
  Therefore the state becomes as follows and the pointer itself is allocated to ROM (.const/.zconst/.zconst23).

```
*p = 0;     /*error*/
p = 0;      /*error*/
```

A.2 Functions

This section explains functions.

A.2.1 Changing area to be allocated to

When changing a program area's section name, specify the function using the #pragma section directive as shown below.

```
#pragma section text  ["section name"]
```

If you create an arbitrary section with a text attribute using the #pragma section directive, the name of the section that is generated will be "specified-string + text".
Specify the start address of the section with the -start option, as follows.

```
-start=sec.text/1000
```

Specify the address as a base-16 number. If the address is not specified, it will be assigned from address 0.

A.2.2 Calling away function

The C compiler uses the jarl instruction to call functions.
However, depending on the program allocation the address may not be able to be resolved, resulting in an error when linking because the jarl instruction is 22-bit displacement.
One way to resolve the error above is to first specify -Xcall_jump=32 to generate jarl32 and jr32 instructions.
If the -Xcall_jump=22 option is specified, then you can make function calls that do not depend on the displacement width by specifying the C compiler's -Xfar_jump option.
When calling a function set as far jump, the jarl32 and jr32 instruction rather than the jarl instruction is output.
One function is described per line in the file where the -Xfar_jump option is specified. The names described should be C language function names prefixed with "_" (an underscore).

Example The file where the -Xfar_jump option is specified

```
_func_led
_func_beep
_func_motor
    :
_func_switch
```

If the following is described in place of "_function-name", all functions will be called using far jump.

```
{all_function}
```

A.2.3 Embedding assembler instructions

With the CC-RH assembler instructions can be described in the following formats within C source programs. This treats the function itself as an 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*/
}
```

See "Describing assembler instruction" for details.

A.2.4 Executing a specific routine from RAM

If you wish to have specific routines in your program be executed from RAM, use the -rom option of the optimizing linker as in the procedure described below.

(1) Allocate sections to addresses in ROM where you wish to place the routines and the corresponding addresses in RAM, respectively.

(2) Prepare a separate routine for transfer that will lead to the following actions and include it in your program.
(2-1) Copy the target routines from the ROM sections to the corresponding RAM sections.
(2-2) Call the target routines from RAM.

(3) In the process of building your program, run the optimizing linker with the following settings for such code.
(3-1) As parameter *ROMsection* of the -rom option, specify the name of the ROM section where the target routine for copying is to be allocated.
(3-2) As parameter *RAMsection* of the -rom option, specify the name of the RAM section from which you wish the target routine to be run.

A.3 Variables (Assembler)

This section explains variables (Assembler).

A.3.1 Defining variables with no initial values

Use the .db 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 define the label with the .public directive.

```
.public label 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  4               ;Aligns _val0 to 4 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/.dhw directives/.db4/.dw directives in the section with the default value.

- 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 define the label with the .public directive.

```
.public label_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  4            ;Aligns _val0 to 4 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 _val0, and stores 10 in it
_val2:
        .db     1            ;Allocates a 1-byte area for _val0, and stores 1 in it
```

A.3.3 Defining const constants

To define a const, use the .db directives/.db2/.dhw directives/.db4/.dw directives within the .const, .zconst or .zconst23 section.

- 1-byte values

```
[label:]        .db value
```

- 2-byte values

```
[label:]        .db2 value
```

- 4-byte values

```
[label:]        .db4 value
```

Example Defining const constants

```
        .cseg   const
        .public _p              ;Sets _p as able to be referenced from other files
        .align  4               ;Aligns _val0 to 4 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	Sep 14, 2015	-	First Edition issued
1.01	Jul 01, 2016	16	The exclusive control check setting file is added.
		24, and others	The -Xcheck_exclusion_control compile option is added.
		46	[Detailed description] is changed.
		57, 58	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
		169	[Detailed description] is changed.
		671, 834	Dynamic memory management functions are added.
		836, 837	The descriptions of the security facility is added.
		840 and others	The description of the startup is changed.
		870 and others	Unnecessary messages are deleted.
		878	E0521158 is added.
		891	F0523089 is added.
		898	W0511143 is added.
		899	W0520171 is added.
1.02	Dec 01, 2016	12	The description of "License" is changed.
		12	"Standard and Professional Editions" is added.
		12	"Free Evaluation Editions" is added.
		20	The operation when a subcommand file is specified is changed.
		22, and others	The -g_line compile option is added.
		23, and others	The -Xuse_fp16 compile option is added.
		57, 59, 61, 250	The __fp16 type is added.

Rev.	Date	Description	
		Page	Summary
1.02	Dec 01, 2016	58, 59	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
		68	[Detailed description] is changed.
		80	[Detailed description] is changed.
		93	[Detailed description] is changed.
		97	[Detailed description] is changed.
		112	The classifications of -D, -U, and -I option are changed.
		162, 175	The operation when a section cannot be allocated is changed.
		235, and others	The descriptions of the following implementation-defined items are changed. 4.1.3 (1), (4), (6), (7), (9), (12), (14), (16), (27), (30), (36), (37), (38)
		238	A description of the half.h file is added.
		247	The descriptions of the types that can represent decimal constants are changed.
		258	The following reserved words are added. __fp16, __set_il_rh, __ldsr_rh, __stsr_rh
		268	The description is changed.
		277	The contents of Remark are changed.
		280	The description of "Embedded Function" is changed.
		281	The return type of __stcw is changed.
		285	The following descriptions are changed. Alignment condition of top structure object Size of structure objects
		289	[Caution] is changed.
		291	Unnecessary descriptions are deleted.
		293	The description is changed.
		295, 296	A description of "Half-precision floating-point type" is added.
		364	[Caution] is changed.
		675	A description of "Other instructions" is changed.
		704	[Description] is changed.
		807-810	[Return value] is changed.
		845	The description is changed.
		854	The description is changed.
		899, 917	E0595001, E0595002, E0595003, E0595004, E0595005, E0595010, and W0595020 are added.
		904	F0563020 and F0563115 are added.

Rev.	Date	Description	
		Page	Summary
1.02	Dec 01, 2016	907	M0560700 is added.
		908	The description of W0511179 is changed.
		908	W0511180, W0511183 are added.
		911	W0523068 is added.
		913	W0561014 is added.
		919	The description of "-Xcpu Option Specification for Assembler" is changed.
		919	"Controlling the Output of Bit Manipulation Instructions" is added.
		920	"Tool for Confirming SYNCP Instruction Insertion at the Beginning of Exception Handler" is added.
		927	The description is changed.
1.03	Jun 01, 2017	59	The following MISRA-C:2012 rules are added. 12.5 13.2 13.5 17.5 17.8 21.13 21.15 21.16
		62	[Detailed description] of the -Xuse_fp16 option was changed.
		24, 76, 101	The -insert_dbtag_with_label option was added.
		24, 76, 102, 261, 298-300	The -store_reg option and #pragma register_group directive were added.
		108, 144	The ranges for the message numbers that can be controlled by the -Xno_warning option were listed.
		110	[Example of use] of the -Xasm_option option was changed.
		158, 159, 180, 181, 214-216	The relocation_attribute specification was added to the -SHow option.
		163	The -END_RECORD option was added.
		172	[Example of use] of the -PADDING option was changed.
		173	[Detailed description] of the -OVERRUN_FETCH option was changed.
		243	The type of the sizeof operator was added.

Rev.	Date	Description	
		Page	Summary
1.03	Jun 01, 2017	245, 260, 261, 269, 271, 273, 275, 281, 284, 293-295	Descriptions on the #pragma directives were changed.
		249	The __fp16 type was added to the description on types that cannot be used when the -Xansi option is specified.
		283	Descriptions on embedded functions __mul32() and __mul32u() were changed.
		371	A description on the .db pseudo-instruction was changed.
		676, 677	A description on the error number that is output when using the cmovf.d or cmovf.s instruction was modified.
		682-684	Special symbols were added.
		687	A description on half.h was added.
		687	Descriptions on reentrancy were changed.
		854-861	A chapter regarding the data sections used by standard library functions and a list of reentrancy was added.
		867	A dynamic processing for determining whether the FPU is available during the initial setup of the FPU in the startup routine was added.
		883	A note regarding allocation of the ROM or RAM section was added.
		893-932	[Explanation] was added for the following messages. C0511200, C0519996, C0519997, C0530001, C0530002, C0530003, C0530004, C0530005, C0530006, C0550802, C0550804, C0550805, C0550806, C0550808, C0551800, C0564001, E0511200, E0523069, E0523070, E0523071, E0523072, E0550270, E0550605, E0550606, E0550633, E0550637, E0550638, E0550639, E0550640, E0550641, E0550647, E0550649, E0551401, E0551402, E0551403, E0551406, E0551501, E0562430, E0562431, E0562432, E0562433, E0562434, E0562435, E0562436, E0562437, E0562438, E0562439, E0562450, E0562451, F0563103, M0536001, W0511184, W0511185, W0523120, W0550010, W0561015, W0561016, W0561017, W0561210
		920	The message of F0563102 is changed.
		903-927	[Explanation] was deleted for the following messages. E0550204, E0550205, E0550206, E0550263, E0550264, E0550266, E0550268, E0550642, E0551228, E0551235, E0551301, E0551307, E0551312, E0551316, F0551602, W0550605, W0550606, W0550645, W0550647, W0550649
		903-929	[Explanation] was changed for the following messages. E0550212, E0550236, E0550237, E0550239, E0550240, F0550511, F0550512, F0550537, F0550539, W0550013, W0550019, W0561004
		945, 946	The comment format of coding examples was changed.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2017	10, 12, 265, 313, 696	The C99 standard is supported.
		23, 50, 51, 57, 227, 265-267, 271	The -lang compile option is added.
		23, 50, 52, 63, 266, 267, 271	The -strict_std compile option is added.
		23, 77, 84	The -r4 compile option is added.
		24, 77, 105-107	The -control_flow_integrity compile option is added.
		24, 77, 108	The -pic compile option is added.
		24, 77, 109	The -pirod compile option is added.
		24, 77, 110	The -pid compile option is added.
		25, 115, 117	The -change_message compile option is added.
		59, 60	The following MISRA-C:2012 rules are added. 8.14 9.4 9.5 13.1 17.6 18.7 21.11 21.12
		67-69	The contents of [Detailed description] of the -O compile option are changed.
		68, 69	The -Oinline_init compile option is added.
		82	The contents of [Detailed description] of the -Xreg_mode compile option is changed.
		85	The contents of [Detailed description] of the -Xep compile option is changed.
		88	The contents of [Detailed description] of the -Xfar_jump compile option is changed.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2017	99	The contents of [Detailed description] of the -Xstack_protector compile option is changed.
		101	The following parameters are added to the -Xsection compile option. pcconst16, pcconst23
		116	The specifiable range of the -Xno_warning compile option is changed.
		124, 144, 148	The -pic assemble option is added.
		124, 144, 149	The -pirod assemble option is added.
		124, 144, 150	The -pid assemble option is added.
		145	The contents of [Detailed description] of the -Xreg_mode assemble option is changed.
		147	The contents of [Detailed description] of the -Xep assemble option is changed.
		156	The description in [Specification format] of the -Xno_warning assemble option is changed.
		160, 169, 171, 185	The -FIX_RECORD_LENGTH_AND_ALIGN link option is added.
		160, 169-171, 192	The -CFI link option is added.
		160, 169-171, 193	The -CFI_ADD_Func link option is added.
		160, 169-171, 194	The -CFI_IGNORE_Module link option is added.
		170	The -nocompress option is added.
		170, 171	The all and total_size specification are added to the -SHow option.
		171	The -crc option is added.
		184	The specifiable condition of the -BYte_count link option is changed.
		197, 198, 231, 236	cfi is added to the parameter of the -SHow link option.
		228	The description of "Relationship with #pragma directives" is changed.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2017	256	The description of "Translation limit" is changed.
		262, 263	The description of "Predefined macro names" is changed.
		275	The description of "Compiler generated symbols" is changed.
		276-282	The description of "Allocation of function and data to section" is changed.
		283, 285, 293, 294, 309	The description of the simultaneous specification of the #pragma directive is changed.
		285	The description of "Functions for which inline expansion should be prevented" is changed.
		306	The description of "Format for specifying core number" is changed.
		309	The description of "Detection of stack smashing" is changed.
		312, 313	The description of "Detection of writing to control registers or insertion of synchronization processing" is changed.
		363, 364	Relocation attributes pctext, pcconst16, pcconst23, and pcconst32 are added to the .cseg directive.
		365-367	Relocation attributes sdata32, sbss32, edata32, and ebss32 are added to the .dseg directive.
		369	The description of [Caution] of the .org directive is changed.
		377	The description of the .stack directive is changed.
		425	The description of "Reserved Words" is changed.
		426	The description of "Predefined Macro Names" is changed.
		560	The description of the jarl instruction is changed.
		693	The description of "Special Symbol" is changed.
		694, 857, 871	A checking function to detect illegal indirect calls is added.
		696	The following header files are added. iso646.h, stdbool.h, and stdint.h
		778, 783, 786, 790, 791	The descriptions of [Caution] of llabs, lldiv, atoll, strtoll, and strtoull are changed.
		841, 842	The descriptions of [Return value] of atan2f and atan2 are changed.
		860	The description of [Example] of the __heap_chk_fail function is changed.
		876	The description of "Stack area" is changed.

Rev.	Date	Description	
		Page	Summary
1.04	Dec 01, 2017	886-888	The description of "Symbols" is changed.
		889-899	The description of "PIC/PID Facility" is added.
		900	The description of "Registers not guaranteed to be same before and after function call (Caller-Save registers)" is changed.
		917, 918, 922, 934-936, 944, 946	The following messages are added. E0520411, E0523087, E0550652, F0563003, F0563150, F0563431, F0563600, F0563601, F0563602, W0561142, and W0561331
		927, 928, 934, 944	The following messages are changed. E0562311, E0562340, E0562417, F0563004, and W0561130
		939	The description of [Explanation] of W0520062 is changed.
		911-946	The following messages are deleted. E0511120, E0562017, E0562021, E0562112, E0562113, E0562142, E0562203, E0562220, E0562223, E0562323, E0562331, E0562402, E0562404, E0562405, E0562500, F0563115, F0563120, F0563311, F0563312, F0563313, F0563400, F0563420, M0560102, M0560103, M0560300, M0560510, M0560511, W0561015, W0561110, W0561180, W0561182, W0561183, W0561190, W0561192, W0561194, W0561321, W0561325, W0561430, W0561500, W0561501, and W0561502
		950	The descriptions of "GP relative access" and "EP relative access" are changed.
		951, 952	The description of "Changing allocated section" is changed.
1.05	Jun 01, 2018	12	The description of "License" is changed.
		18	The description of "Specification format" is changed.
		23, 66, 77	The -library compile option is added.
		24, 78, 88	The -Xfxu compile option is added.
		24, 78, 95	The -use_recipf compile option is added.
		24, 78, 96	The -relaxed_math compile option is added.
		24, 78, 107	The -Xresbank_mode compile option is added.
		38	The description of the -Xcommon compile option is changed.

Rev.	Date	Description	
		Page	Summary
1.05	Jun 01, 2018	39	The description of the -Xcpu compile option is changed.
		51	The description of the -lang compile option is changed.
		90, 91	The description of the -Xfar_jump compile option is changed.
		121	The description of the -Xno_warning option is changed.
		122	The description of the -change_message option is changed.
		140	The description of the -Xcommon assemble option is changed.
		141	The description of the -Xcpu assemble option is changed.
		161	The description of the -Xno_warning assemble option is changed.
		168	The description of the -Input link option is changed.
		171	The description of the -Binary link option is changed.
		172	The description of the -DEFine link option is changed.
		175	The description of the -FOrm link option is changed.
		180	The description of the -RECord link option is changed.
		181	The description of the -END_RECORD link option is changed.
		183	The description of the -OUtput link option is changed.
		187	The description of the -NOMessage link option is changed.
		165, 174, 195, 196	The description of the -CRc link option is changed.
		198	The description of the -CFI_ADD_Func link option is changed.
		199	The description of the -CFI_IGNORE_Module link option is changed.
		205	The description of the -STARt link option is changed.
		210	The description of the -CPu link option is changed.
		219	The description of the -RENname link option is changed.
		220	The description of the -DELete link option is changed.
		221	The description of the -REPlace link option is changed.
		224	The description of the -CHange_message link option is changed.
		262, 263	"The limit values of the integer types (stdint.h file)" is added.
		294	The description of "Enable or disable acknowledgement of maskable interrupts (interrupt mask)" is changed.
		294-297, 301, 302	The description of "Describing interrupt/exception handler" is changed.

Rev.	Date	Description	
		Page	Summary
1.05	Jun 01, 2018	304-307	The description of "Embedded functions" is changed.
		306, 307	The following embedded functions are added. __dbcp(), __dbpush(), __dbtag(), __clipb(), __clipbu(), __cliph(), __cliphu(), __ldlbu(), __ldlhu(), __stcb(), and __stch()
		316, 317	The description of "Format for specifying core number" is changed.
		695	The description of cmpf.s is changed.
		814-839	The description of "Mathematical functions" is changed.
		814, 837-839, 860	The following functions are added. fmax, fmaxf, fmin, fminf, copysign, and copysignf
		902-934	The following messages are added. E0511133, E0511182, E0520069, E0520117, E0520175, E0520296, E0520393, E0520404, E0520469, E0520643, E0520644, E0520654, E0520655, E0520702, E0520749, E0520757, E0520765, E0520938, E0520965, E0520966, E0520967, E0520968, E0520969, E0520976, E0520977, E0521029, E0521030, E0521031, E0521037, E0521038, E0521039, E0521040, E0521045, E0521049, E0521051, E0521052, E0521144, E0521158, E0521260, E0521261, E0521649, E0523026, E0523027, E0523048, E0523065, E0523067, E0523073, E0523090, E0523091, E0523118, E0523119, E0523122, E0523123, E0523124, E0523125, E0523126, E0523127, F0520163, F0520164, F0520182, F0520571, F0520642, F0520920, F0523073, W0511181, W0520055, W0520083, W0520140, W0520159, W0520220, W0520221, W0520222, W0520223, W0520240, W0520257, W0520513, W0520609, W0520660, W0520767, W0520819, W0520867, W0520940, W0520951, W0520966, W0520967, W0520968, W0521037, W0521039, W0521040, W0521046, W0521051, W0521057, W0521072, W0521222, W0521223, W0521224, W0521273, W0521297, W0523038, and W0523116
		902-934	The following messages are deleted. E0511118, E0511136, E0511142, E0511148, E0511161, E0520001, E0520002, E0520005, E0520096, E0520123, E0520126, E0520157, E0520170, E0520255, E0520257, E0520259, E0520518, E0520544, E0520545, E0520606, E0520661, E0520668, E0520767, E0520940, E0520989, E0520992, E0520993, E0521066, E0521075, E0521076, E0521254, E0521255, E0521282, E0521420, E0523042, E0523059, F0512003, F0520016, F0520219, F0520583, F0520584, F0523071, W0520001, W0520014, W0520144, W0520171, W0520181, W0520185, W0520224, W0520225, W0520226, W0520514, W0520902, W0521396, W0523060, and W0523120
		942	"Version of Compiler Package" is added.

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2018	13	The descriptions in "(3) Optimizing linker (rlink)" are changed.
		23, 67, 79, 130, 144, 145, 170, 177, 179, 180, 207, 209-213, 216, 228, 236	The following options are added for link-time optimization. Compile option: -goptimize Assemble option: -goptimize Link options: -OPtimize/-NOOPtimize, -SEction_forbid, -Absolute_forbid, and -SYmbol_forbid
		23, 50, 63	The compile option -misra_intermodule is added.
		59	The following MISRA-C:2012 rules are added. 8.5 8.6
		170, 223, 230	The link option -LIB_REName is added.
		179, 180	[Detailed description] of the link option -FOrm is changed.
		229	[Detailed description] of the link option -REName is changed.
		238	[Remark] of the link option -Total_size is changed.
		269-294	The configuration of "4.1 Basic Language Specifications" was reviewed.
		300-302	The configuration of "4.2 Extended Language Specifications" was reviewed.
		344-346	The descriptions in "4.2.6.14 Detection of writing to control registers or insertion of synchronization processing" are changed.
		468-505	"5.9 Description of Instructions" is deleted and "5.9 Extension of Assembly Language" is added.
		619-647	The following library functions are added. acosl, asinl, atanl, atan2l, cosl, sinl, tanl, coshl, sinhl, tanhl, expl, frexpl, ldexpl, logl, log10l, modfl, fabsl, powl, sqrtl, ceill, floorl, round, roundf, roundl, lround, lroundf, lroundl, llround, llroundf, llroundl, trunc, truncf, truncl, fmodl, copysignl, fmaxl, and fminl
		637, 644	[Detailed description] of pow functions and fmod functions are changed.
		675, 676	The descriptions in "(4) Initialization of RAM sections" are changed.
		686, 687	"8.5 ROMization" is deleted and "8.5 Creating ROM Images" is added.

Rev.	Date	Description	
		Page	Summary
1.06	Dec 01, 2018	725, 734, 745	The description of [Explanation] of E0562212, E0562320, F0563004, and W0561301 are changed.
		728, 736, 738	The following messages are added. E0562600, M0560004, M0560005, M0560100, and W0520070
		743	The message of W0561101 is changed.
		757	"A.2.4 Executing in RAM" is deleted and "A.2.4 Executing a specific routine from RAM" is added.
1.07	Nov 01, 2019	10	The description of "Copyrights" is changed.
		18	The description of "Example of operations" is changed.
		58, 59	The following MISRA-C:2012 rules are added: 8.13 14.2 14.3
		88	The description of "Interpretation when omitted" and [Detailed description] of the -Xfloat compile option are changed.
		23, 79, 95	The description of the -relaxed_math compile option is moved.
		95	The description of the -relaxed_math compile option is entirely changed.
		23, 79, 98	The -approximate compile option is added.
		123	[Detailed description] of the -Xno_warning compile option is changed.
		168, 171, 178, 181, 229	The -ALLOW_DUPLICATE_MODULE_NAME link option is added.
		194	[Remark] of the -BYte_count link option is changed.
		230	[Specification format] and [Example of use] of the -LIB_RENName link option are changed.
		288	The following item in "Implementation-defined behavior of C99" is changed: (109)
		294, 295	The headers of the following tables are changed: Table 4.2, Table 4.3, Table 4.4
		296	The description of "Internal representation" in "Floating-point type" is changed.
		301	Figure 4.8 is added (restored).
		303	The description of the macro name MULTI_LEVEL in Figure 4.8 is changed.
		312	The description of the effect of the #pragma section directive is changed.
		353	The description of "Modification of C source" is changed.

Rev.	Date	Description	
		Page	Summary
1.07	Nov 01, 2019	475 and others	In "Extension of Assembly Language", a header line is added to some tables.
		522, 523	In Table 7.1, the "Supplied Libraries" column is changed.
		524	The following row is deleted from Table 7.2: inttypes.h
		678 and others	The table in "Usage of Data Sections and List of Reentrancy" is changed.
1.08	Nov 01, 2020	Front cover	The target CPU cores are added.
		15	The tool usage information file is added to Table 2.1.
		23, 80, 109	The -stuff compile option is added.
		47	[Detailed description] for the -Xpreinclude compile option is changed.
		69	The align optimization item is added.
		100	[Detailed description] for the -Xunordered_cmpf compile option is changed.
		172, 226, 242	The -VERBOSE link option is added.
		305	The description in "Reserved words" is changed.
		309, 333 and others	Wording "embedded function" is changed to "intrinsic function".
		313, 314	The format description of the #pragma section directive is changed.
		314	Table 4.16 is divided into Tables 4.16 to 4.18.
		333-335	The format of the function declarations in the following tables is changed to ANSI-C: Tables 4.20 and 4.21
		411	[Syntax] and [Use] of the .section directive are changed.
		736	The following message is added: E0550271

Rev.	Date	Description	
		Page	Summary
1.09	Nov 01, 2021	10	The description of "GENERAL" is changed.
		17	The description of "Specification format" in "Command line operation" is changed.
		23, 80, 82	The -misalign compile option is added.
		46	[Detailed description] of the -I compile option is changed.
		68, 69	The descriptions of the following optimization items are changed: tail_call, align
		150	[Detailed description] of the -D assemble option is changed.
		153	[Detailed description] of the -I assemble option is changed.
		309	The description of "#pragma directive" is changed.
		325, 326	A note is added in the table for "interrupt specification".
		327-329, 331	The order of the ldsr instruction is changed in the restoration processing of the exception handler.
		360, 404	$ is added to identifiers.
		418, 424	The dbl_size directive is added.
		435, 438	The description of the .extern directive is changed.
		527	The description of "Special Symbol" is changed.
		724	An error in the example is corrected in "Reference of Argument Defined by Other Language".
		728, 754, 756, 762	The following messages are changed: E0511178, F0563430, W0511180, W0511185, W0561016, W0561017
		732, 759	Errors in the following messages are corrected: E0520137, W0521053

Rev.	Date	Description	
		Page	Summary
1.10	Dec 01, 2022	67	The description in "Interpretation when omitted" for the -O compile option is changed.
		103	[Detailed description] for the -Xpatch compile option is changed.
		150	[Detailed description] for the -D assemble option is changed.
		172, 182, 203	The -RESERVE_PREFETCH_AREA link option is added.
		204, 205	[Detailed description] and [Example of use] for the -CRc link option are changed.
		531	Table 7.2 is added.
		747, 755	The following messages are added: E0562326 and F0563115
		751, 764	The following messages are changed: F0520571 and W0561017
1.11	Dec 01, 2023	16	The description of "Command line operation" is changed.
		20, 39	The description of the -P compile option and its [Detailed description] and [Example of use] are changed.
		61	[Remarks] of the -misra_intermodule compile option is changed.
		102	[Detailed description] of the -Xpatch compile option is changed.
		172, 213, 218	The -ALLOW_OPTIMIZE_ENTRY_BLOCK link option is added.
		190	[Detailed description] of the -ROm link option is changed.
		205	[Detailed description] of the -CRc link option is changed.
		531	Table 7.2 is changed.
		637	[Caution] of the setjmp function is changed.
		727	The following message is changed: C0519996
		727, 744	The following messages are added: C0520000, C0529000, and E0562114

CC-RH User's Manual

Publication Date: Rev.1.00 Sep 14, 2015
Rev.1.11 Dec 01, 2023

Published by: Renesas Electronics Corporation

CC-RH

Renesas Electronics Corporation

R20UT3516EJ0111