User's Manual

CC-RH V1.02.00

RH850 Family C Compiler Package

User's Manual: Compiler

All information contained in these matenals, 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.1.00 2015.02

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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended 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; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

All trademarks or registered trademarks in this document are the property of their respective owners.

TABLE OF CONTENTS

1. GENERAL 10
1.1 Outline 10
1.2 Special Features 10
1.3 Limits 10
2. COMMAND REFERENCE 11
2.1 Overview 11
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 102
2.5.3 Link options 134
2.6 Specifying Multiple Options 193
2.6.1 Priority 193
2.6.2 Incompatible features 193
2.6.3 Dependencies 193
2.6.4 Relationship with #pragma directives 194
3. OUTPUT FILES 195
3.1 Assemble List File 195
3.1.1 Structure of the assemble list 195
3.1.2 Assemble list 195
3.1.3 Section list 196
3.1.4 Command line information 196
3.2 Link Map File 197
3.2.1 Structure of link map 197
3.2.2 Option information 197
3.2.3 Error information 198
3.2.4 Link map information 198
3.2.5 Total section size 199
3.2.6 Symbol information 199
3.2.7 Cross reference information 201
3.3 Link Map File (When Objects Are Combined) 203
3.3.1 Structure of link map 203

3.3.2 Header information . . . 203
3.3.3 Option information . . . 203
3.3.4 Error information . . . 204
3.3.5 Entry information . . . 204
3.3.6 Combined address information . . . 204
3.3.7 Address overlap information . . . 205
3.4 Library List File . . . 206
3.4.1 Structure of the library list . . . 206
3.4.2 Option information . . . 206
3.4.3 Error information . . . 207
3.4.4 Library information . . . 207
3.4.5 Module, section, and symbol information within the library . . . 208
3.5 Intel HEX File . . . 209
3.5.1 Structure of the Intel HEX file . . . 209
3.5.2 Start linear address record . . . 210
3.5.3 Expanded linear address record . . . 210
3.5.4 Start address record . . . 211
3.5.5 Expanded address record . . . 211
3.5.6 Data record . . . 212
3.5.7 End record . . . 212
3.6 Motorola S-record File . . . 214
3.6.1 Structure of the Motorola S-record file . . . 214
3.6.2 S0 record . . . 215
3.6.3 S1 record . . . 215
3.6.4 S2 record . . . 216
3.6.5 S3 record . . . 216
3.6.6 S7 record . . . 216
3.6.7 S8 record . . . 217
3.6.8 S9 record . . . 217

4. COMPILER LANGUAGE SPECIFICATIONS . . . 218
4.1 Basic Language Specifications . . . 218
4.1.1 Unspecified behavior . . . 218
4.1.2 Undefined behavior . . . 218
4.1.3 Processing system dependent items . . . 220
4.1.4 C99 language function . . . 228
4.1.5 Option to process in strict accordance with ANSI standard . . . 230
4.1.6 Internal representation and value area of data . . . 231
4.1.7 Section name . . . 236
4.1.8 Register mode . . . 237
4.2 Extended Language Specifications . . . 239
4.2.1 Macro name . . . 239

4.2.2 Reserved words . . . 240
4.2.3 Compiler generated symbols . . . 240
4.2.4 #pragma directive . . . 240
4.2.5 Using extended language specifications . . . 242
4.2.5.1 Allocation of data and program to section . . . 242
4.2.5.2 Describing assembler instruction. . . . 248
4.2.5.3 Inline expansion. . . . 249
4.2.5.4 Controlling interrupt level. . . . 252
4.2.5.5 Interrupt/Exception processing handler . . . 252
4.2.5.6 Disabling or enabling maskable interrupts. . . . 258
4.2.5.7 Embedded functions . . . 259
4.2.5.8 Structure type packing . . . 262
4.2.5.9 Bit field assignment . . . 269
4.2.5.10 Core number specification (for a multi-core device) . . . 270
4.2.6 Modification of C source . . . 273

5. ASSEMBLY LANGUAGE SPECIFICATIONS. . . . 274

5.1 Description of Source. . . . 274
5.1.1 Description . . . 274
5.1.2 Expressions and operators . . . 280
5.1.3 Arithmetic operators . . . 283
5.1.4 Logic operators . . . 291
5.1.5 Relational operators . . . 296
5.1.6 Shift operators . . . 305
5.1.7 Byte separation operators. . . . 308
5.1.8 2-byte separation operators . . . 311
5.1.9 Section aggregation operators . . . 315
5.1.10 Other operator . . . 320
5.1.11 Restrictions on operations . . . 322
5.1.12 Identifiers . . . 323
5.2 Directives . . . 324
5.2.1 Outline . . . 324
5.2.2 Section definition directives . . . 325
5.2.3 Symbol definition directives . . . 335
5.2.4 Compiler output directives . . . 338
5.2.5 Data definition/Area reservation directives . . . 344
5.2.6 External definition/External reference directives. . . . 355
5.2.7 Macro directives . . . 359
5.3 Control Instructions . . . 370
5.3.1 Outline . . . 370
5.3.2 Assembler control instructions . . . 371
5.3.3 File input control instructions . . . 379

5.3.4 Conditional assembly control instructions . . . 382
5.4 Macro . . . 391
5.4.1 Outline . . . 391
5.4.2 Usage of macro . . . 391
5.4.3 Macro operator . . . 391
5.5 Reserved Words . . . 393
5.6 Predefined Macro Names . . . 393
5.7 Assembler Generated Symbols . . . 393
5.8 Instruction Set . . . 394
5.9 Description of Instructions . . . 403
5.9.1 Load/Store instructions . . . 405
5.9.2 Arithmetic operation instructions . . . 417
5.9.3 Saturated operation instructions . . . 469
5.9.4 Logical instructions . . . 478
5.9.5 Branch instructions . . . 513
5.9.6 Bit manipulation instructions . . . 534
5.9.7 Stack manipulation instructions . . . 543
5.9.8 Special instructions . . . 548
5.9.9 Loop instructions . . . 581
5.9.10 Floating-point operation instructions . . . 584
5.9.11 Other instructions . . . 658

6. SECTION SPECIFICATIONS . . . 659
6.1 Sections . . . 659
6.1.1 Section concatenation . . . 659
6.2 Special Symbol . . . 661

7. LIBRARY FUNCTIONAL SPECIFICATIONS . . . 663
7.1 Supplied Libraries . . . 663
7.2 Header Files . . . 663
7.3 Re-entrant . . . 664
7.4 Library Function . . . 665
7.4.1 Program diagnostic functions . . . 665
7.4.2 Functions with variable arguments . . . 667
7.4.3 Character string functions . . . 671
7.4.4 Memory management functions . . . 687
7.4.5 Character conversion functions . . . 693
7.4.6 Character classification functions . . . 696
7.4.7 Standard I/O functions . . . 709
7.4.8 Standard utility functions . . . 743
7.4.9 Non-local jump functions . . . 772
7.4.10 Mathematical functions . . . 776
7.4.11 RAM section initialization function . . . 822

7.4.12 Peripheral device initialization function . . . 825
7.4.13 Operation runtime functions . . . 827

8. STARTUP . . . 830
8.1 Outline . . . 830
8.2 Startup Routine . . . 830
8.2.1 Exception vector table . . . 830
8.2.2 Startup routine for the boot loader project . . . 831
8.2.3 Startup routine for the application project . . . 833
8.2.4 Passing information from the application project to the boot loader project . . . 837
8.3 Coding Example . . . 838
8.4 Symbols . . . 849
8.4.1 Global pointer (gp) . . . 849
8.4.2 Element pointer (ep) . . . 852
8.5 ROMization . . . 855
8.5.1 Outline . . . 855
8.5.2 Creating ROMized load module file . . . 855

9. FUNCTION CALL INTERFACE SPECIFICATIONS . . . 857
9.1 Function Call Interface . . . 857
9.1.1 General-purpose registers guaranteed before and after function calls . . . 857
9.1.2 Setting and referencing arguments and return values . . . 857
9.1.3 Address indicating stack pointer . . . 859
9.1.4 Stack frame . . . 859
9.2 Calling of Assembly Language Routine from C Language . . . 862
9.3 Calling of C Language Routine from Assembly Language . . . 863
9.4 Reference of Argument Defined by Other Language . . . 863
9.5 General-purpose Registers . . . 864

10. MESSAGE . . . 865
10.1 General . . . 865
10.2 Message Formats . . . 865
10.3 Message Types . . . 865
10.4 Message Numbers . . . 865
10.5 Messages . . . 865
10.5.1 Internal errors . . . 866
10.5.2 Fatal errors . . . 868
10.5.3 Abort errors . . . 920
10.5.4 Information . . . 937
10.5.5 Warnings . . . 940

11. CAUTIONS . . . 970
11.1 Volatile Qualifier . . . 970

11.2 V850E3v5 G3K Core Specification for Assembler (-Xcpu=g3k Option) 971

A. QUICK GUIDE 972

A.1 Variables (C Language) 972

A.1.1 Allocating to sections accessible with short instructions 972

A.1.1.1 GP relative access 972

A.1.1.2 EP relative access 972

A.1.2 Changing allocated section 973

A.1.3 Change the allocated area using the -Xpreinclude option 975

A.1.4 Defining variables for use during standard and interrupt processing 975

A.1.5 Defining const constant pointer 976

A.2 Functions 977

A.2.1 Changing area to be allocated to 977

A.2.2 Calling away function 977

A.2.3 Embedding assembler instructions 978

A.2.4 Executing in RAM 978

A.3 Variables (Assembler) 978

A.3.1 Defining variables with no initial values 978

A.3.2 Defining variable with initial values 979

A.3.3 Defining const constants 979

1. GENERAL

This chapter provides a general outline of the RH850 family's C compiler package (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 ANSI standard
The C language specifications conform to the ANSI standard.

(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 format is used for debugging information.

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

1.3 Limits

The maximum values that can be coded in C and assembly-language programs are indicated below.

(1) Compiler limits
There are no limits on translation. The maximum translatable value depends on the memory of the host machine on which the program is running. Exceeding the maximum value an assembler specified will result in an error.

Remark The maximum number of bytes for one object is 2 Gbytes (in the host environment).

(2) Assembler limits
The maximum values that can be coded in assembly-language programs are indicated below.

Table 1.1 Assembler Limits

Description	Limit
Symbol length (Token length)	4,294,967,294[Note]
Label length (Token length)	4,294,967,294[Note]
Number of symbols	4,294,967,294[Note]
Number of parameters in LOCAL directive	4,294,967,294[Note]
Number of automatically generated LOCAL directive symbols	4,294,967,294[Note]
Nesting levels in INCLUDE control instruction	4,294,967,294 [Note]
TDATA relocation attribute section	256 bytes
ALIGN directive	Even number of 2 or more, but less than 2^{31}
Number of arguments in IRP directive	4,294,967,294[Note]

Note Depends on memory of host machine on which it is running.

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
asrh: Optimizing linker start command

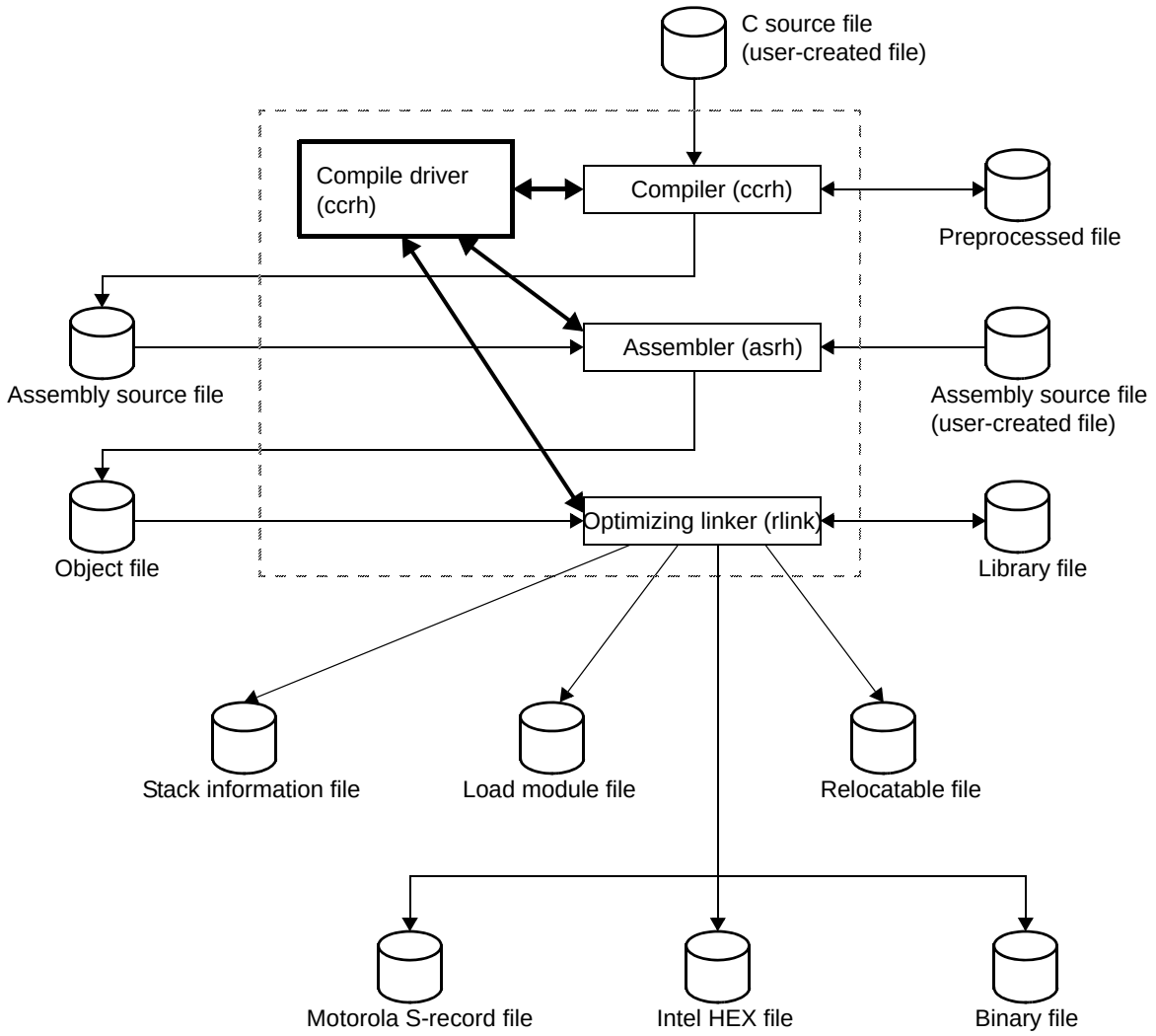
Processing of each command is shown below.

(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.
It also performs ROMization, performs optimization during linking of relocatable files, creates and edits library files, and converts 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.
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.

File Type	Extension	I/O	Description
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.
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.

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 proceeding [ ] can be repeated
{ }: Select from items delimited by the pipe symbol ("|")
Δ: One or more spaces

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 Windows can be specified.
  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.
- The length that can be specified for a file name depends on Windows (up to 259 characters).
- Uppercase characters and lowercase characters are not distinguished for the alphabet of a file name.
- 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.
Assemble list file "sample.prn" are also output.
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=sample.prn -
Xlk_option=-list -osample.abs -Xcommon=v850e3v5
```

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.
Assemble list file "sample.prn" are also output.

```
>ccrh -c file1.c file2.asm -Xasm_option=-Xprn_path=sample.prn -Xcommon=v850e3v5
```

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=v850e3v5
```

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

```
>asrh -c file1.asm file2.asm -Xprn_path=sample.prn -Xcommon=v850e3v5
```

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.
- A subcommand file cannot be nested.
- 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=v850e3v5
-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=v850e3v5 -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.
- 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.
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 executes only preprocess processing for the input 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	-Xansi	This option processes as making C source program comply strictly with the ANSI 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	This option checks source code against the MISRA-C: 2004 rules.
	-Xignore_files_misra	This option specifies files that will not be checked against the MISRA-C: 2004 rules.
	-Xcheck_language_extention	This option enables the source-code checking of the MISRA-C:2004 rules, which are partially suppressed by language extensions.
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.

Classification	Option	Description
Generated code control	-Xpack	This option performs the structure packing.
	-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.
	-Xep	This option specifies how to handle the ep register.
	-Xfloat	This option controls generating floating-point calculation 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.
	-Xuse_fmaf	This option generates product-sum operation instructions.
	-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 the start addresses of functions
	-Xsection	This option specifies the default sections for data
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.
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=v850e3v5
```

-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=v850e3v5
```

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=v850e3v5 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=v850e3v5 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=v850e3v5 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=v850e3v5 main.c
```

Source debugging control

The source debugging control option is 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.
- 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=v850e3v5 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
  An error will occur.
  However, an error will not occur if the -V, -h, or -P option is specified.
  The interpretation when this option is omitted will change as follows in CC-RH V1.02.00 and later versions.
  The instruction set architecture is set to V850E3v5 (this is the same result as when -Xcommon=v850e3v5 is specified).

[Detailed description]

- This option specifies that an object file common to the various devices is generated.
- When this option is specified, only instructions are used in the instruction set architecture of the target, and common magic number *series* that supports the instruction set architecture is embed into the object file.
- The item that can be specified as *series* is shown below.
  An abort error will occur if any other item is specified.

v850e3v5	It is possible to link to the model with the V850E3V5 instruction set architecture, which is specified as a target device.
rh850	The result is the same as when "v850e3v5" is specified.

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

[Example of use]

- To embed the magic number common to models with the instruction set architectures superior to V850E3V5 into the object file to be generated, describe as:

```
>ccrh -Xcommon=v850e3v5 -c main.c
```

-Xcpu

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

[Specification format]

```
-Xcpu=core
```

- Interpretation when omitted
  If the -Xcommon option is specified, 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.
  An abort error will occur if any other item is specified.

g^3m	Generates an object for G3M.
g^3k	Generates an object for G3K.
g^3mh	Generates an object for G3MH.

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

[Example of use]

- To generate an object for G3M, describe as:

```
>ccrh -Xcpu=g3m -c -Xcommon=v850e3v5 main.c
```

Processing interrupt specification

The processing interrupt specification options are as follows.

- -P
- -S
- -c

-P

This option executes only preprocessing for the input file.

[Specification format]

```
-P
```

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

[Detailed description]

- This option executes only preprocessing for the input file and outputs 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 file and output the results to file "main.i", describe as:

```
>ccrh -P -Xcommon=v850e3v5 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=v850e3v5 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=v850e3v5 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=v850e3v5 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=v850e3v5 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

(c) Standard include file folder[Note]

Note *product install folder*\CS+\ccrh\V*x.xx*\inc

(2) $INCLUDE/$BINCLUDE

(a) Path specified by the -I option

(b) Folder with source file

(c) Folder of original C source file

(d) 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=v850e3v5 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*.

[Example of use]

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

```
>ccrh main.c -Xpreinclude=sample.h -Xcommon=v850e3v5
```

-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=v850e3v5 main.c
```

The following example is equivalent to the example above.

```
>ccrh -Xpreprocess=comment -Xpreprocess=line -P -Xcommon=v850e3v5 main.c
```

C language control

The C language control options are as follows.

- -Xansi
- -Xenum_type
- -Xvolatile
- -Xcheck
- -Xmisra2004
- -Xignore_files_misra
- -Xcheck_language_extention

-Xansi

This option processes as making C source program comply strictly with the ANSI standard.

[Specification format]

```
-Xansi
```

- Interpretation when omitted
  Compatibility with the conventional C language specifications is conferred and processing continues after warning is output.

[Detailed description]

- This option processes as making C source program comply strictly with the ANSI standard[Note] and outputs an error or warning for a specification that violates the standard.
- When this option is specified, macro name "__STDC__" is defined as the macro that the value is 1.
- Processing when compiling in strict adherence to the language specification is as follows.
  - Bit fields
    An error will occur if a type other than an 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.

Note This is a standard specified by ISO/IE C9899:1990 (C90).
Although ccrh also accepts some of the specifications added by ISO/IE C9899:1999 (C99), if this option is specified, code in violation of the standard will cause an error.

[Example of use]

- To process as making C source program comply strictly with the ANSI standard and output an error or warning for a specification that violates the standard, describe as:

```
>ccrh -Xansi -Xcommon=v850e3v5 main.c
```

-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=v850e3v5 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=v850e3v5 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 "CS+ Integrated Development Environment User's Manual: RH850 Coding" 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=v850e3v5 main.c
```

-Xmisra2004

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*.
- 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.

[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=v850e3v5 main.c
```

-Xignore_files_misra

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

[Specification format]

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

- Interpretation when omitted
  All C source files are checked.

[Detailed description]

- This option does not check file *file* against the MISRA-C: 2004 rules.
- This option is valid only when the -Xmisra2004 option is specified.
  If the -Xmisra2004 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=v850e3v5 main.c
```

-Xcheck_language_extention

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

[Detailed description]

- This option enables the source-code checks of the MISRA-C:2004 rules in the following cases where they are suppressed by the unique language specifications extended from the C language standard.
  - When the function has no prototype declaration (rule 8.1) and #pragma interrupt is specified for it.
- This option is valid only when the -Xmisra2004 option is specified.
  If the -Xmisra2004 option is not specified, this option will be ignored (A warning will not be output).

[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=v850e3v5 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=v850e3v5 main.c
```

Optimization specification

The optimization specification options are as follows.

- -O
- -Xintermodule
- -Xinline_strcpy
- -Xmerge_string
- -Xalias
- -Xmerge_files
- -Xwhole_program

-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
  Only 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* is 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 When 0 or 1 is specified, the expansion is suppressed. If 2 or more is specified, the number of executions is converted into *N* loops (*N* is a constant), and a loop including code expanded *value* times. If the code size after expansion is too great or if the number of times of execution of the loop is too few, the number of times of expansion may decrease, or the loop may not be expanded at all. In addition, a loop having a complicated structure, such as having inner loops, may not be expanded. 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).
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. This item is always valid when the -Ospeed option is specified.
tail_call	on or off	Converting function calls at the end of functions to jr If "on" is specified, then if there is a function call at the end of a function, and certain conditions are met, a jr instruction will be generated for that call rather than a jarl instruction. 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.
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.

Optimization Item (item)	Parameter (value)	Description
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.

- If *item* is omitted, it is assumed that this option has specified "-Osize".
- If this option is specified more than once for the same *item*, the option specified last will be valid.
- 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
delete_static_func	off	on	on	on
pipeline	off	off	off	on
tail_call	off	off	on	on
map	-	-	-	-
smap	-	-	-	-

[Example of use]

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

```
>ccrh -Osize -Xcommon=v850e3v5 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 assigned directly because "a" and "b"
point to different types.
    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 -Xwhole_program -Xcommon=v850e3v5 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=v850e3v5 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 sconst (const) 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=v850e3v5 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=v850e3v5 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=v850e3v5 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=v850e3v5 main.c
```

Generated code control

The generated code control options are as follows.

- -Xpack
- -Xbit_order
- -Xpass_source
- -Xswitch
- -Xreg_mode
- -Xreserve_r2
- -Xep
- -Xfloat
- -Xcall_jump
- -Xfar_jump
- -Xdiv
- -Xcheck_div_ov
- -Xuse_fmaf
- -Xunordered_cmpf
- -Xmulti_level
- -Xpatch
- -Xdbl_size
- -Xround
- -Xalign4
- -Xsection

-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=v850e3v5 main.c
```

-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=v850e3v5 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=v850e3v5 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 the same format as the if-else statement along a string of case statements. Select this item if the case statements are written in the order of frequency or if only a few labels are used.
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=v850e3v5 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.
- An abort error will occur if the register that cannot be used for the assembly source file.
- The same register mode must be specified for all source files.
  The different register mode cannot be specified for each source file.
  If there are object files with different register modes, an error will occur during linking.
- By specifying this option, the register mode of the software register bank function can be changed.

[Example of use]

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

```
>ccrh -Xreg_mode=22 -Xcommon=v850e3v5 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=v850e3v5 main.c
```

-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 any of the following attribute strings in a #pragma section in the project is used. ep_auto, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23
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.

[Example of use]

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

```
>ccrh -Xep=fix -Xcommon=v850e3v5 main.c
```

-Xfloat

This option controls generating floating-point calculation instructions.

[Specification format]

```
-Xfloat=type
```

- Interpretation when omitted
  If the -Xcpu=g3m option is specified and the -Xcpu option is not specified, floating-point calculation instructions are generated.
  If the -Xcpu=g3k option is specified, runtime function call instructions are generated.

[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 library call instructions for floating-point calculations. Additionally, the FPU math support library will not be linked.
fpu	Generates floating-point calculation instructions of FPU (floating-point unit) for floating-point calculations. Additionally, the FPU math support library will be linked.

- 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=v850e3v5 main.c
```

-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=v850e3v5 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 the jarl32 and jr32 instruction for the branch to functions (including interrupt functions described by C language) 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 during linking if the function is in the range that cannot be branched to by the jarl or jr directive (±2MB or more) 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.
{all interrupt}	All interrupt functions are called.

- Rutime routines can be specified instead of functions.
  In this case, describe the runtime routine name as-is as the function name.
- 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
```

[Example of use]

- To generate the code that uses jmp directive to branch to the function specified in func.fjp, describe as:

```
>ccrh -Xfar_jump=func.fjp -Xcommon=v850e3v5 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=v850e3v5 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=v850e3v5 main.c
```

-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=v850e3v5 main.c
```

-Xunordered_cmpf

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

[Specification format]

```
-Xunordered_cmpf
```

- Interpretation when omitted
  Invalid operation exceptions in floating-point comparison are not detected.

[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 not-a-number in floating-point comparison.

[Example of use]

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

```
>ccrh -Xunordered_cmpf -Xcommon=v850e3v5 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=v850e3v5 file1.c file2.c
```

-Xpatch

This option applies a patch.

[Specification format]

```
-Xpatch=string
```

- Interpretation when omitted
  The patch is not applied.

[Detailed description]

- "dw_access" can be specified as *string*.
  An error will occur if any other item is specified.
- When the -Xpatch=dw_access option is specified, a code is generated without using the ld.dw and st.dw instructions.

[Example of use]

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

```
>ccrh -Xpatch=dw_access -Xcommon=v850e3v5 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 the start addresses of functions.

[Specification format]

```
-Xalign4
```

- Interpretation when omitted
  The alignment value for function start addresses is set to 2.

[Detailed description]

- This option sets the alignment value for function start addresses to 4.
- 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.

-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

- When the attribute is changed by #pragma section, the attribute specified by #pragma section will be valid.
  When ep_disp16 or ep_disp23 is specified for *value*, -Xep=fix is assumed for compilation. If -Omap or -Osmap is used together with this setting, 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=v850e3v5 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=v850e3v5 main.c
```

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[,num]...
-Xno_warning=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.

[Example of use]

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

```
>ccrh -Xno_warning=11146,11147 -Xcommon=v850e3v5 main.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.
- Use this option to pass a file to the assembler containing an identifier that the ccrh driver does not recognize as an assembly source file.
- 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 -Xasm_far_jump option to the assembler, describe as:

```
>ccrh -Xasm_option=-Xasm_far_jump -Xcommon=v850e3v5 main.c
```

The example above has the same meaning as the following.

```
>ccrh -S -Xcommon=v850e3v5 main.c
>asrh -Xasm_far_jump -Xcommon=v850e3v5 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=v850e3v5 main.c
```

The example above has the same meaning as the following.

```
>ccrh -c -Xcommon=v850e3v5 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=v850e3v5
```

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.
Preprocessor control	-D	This option defines assembler symbols.
	-U	This option deletes the assembler symbol definition by the -D option.
	-I	This option specifies the folder to search include files.
Japanese/Chinese character control	-Xcharacter_set	This option specifies the Japanese/Chinese character code.
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.
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.

Classification	Option	Description
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=v850e3v5
```

-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=v850e3v5
```

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 fatal 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=v850e3v5 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=v850e3v5 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=v850e3v5 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=v850e3v5 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
  An error will occur.
  However, an error will not occur if the -V or -h option is specified.

[Detailed description]

- This option specifies that an object file common to the various devices is generated.
- When this option is specified, only instructions are used in the instruction set architecture of the target, and common magic number *series* that supports the instruction set architecture is embed into the object file.
- The item that can be specified as *series* is shown below.
  An abort error will occur if any other item is specified.

v850e3v5	It is possible to link to the model with the V850E3v5 instruction set architecture, which is specified as a target device.
rh850	The result is the same as when "v850e3v5" is specified.

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

[Example of use]

- To embed the magic number common to models with the instruction set architectures superior to V850E3V5 into the object file to be generated, describe as:

```
>asrh -Xcommon=v850e3v5 -c main.asm
```

-Xcpu

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

[Specification format]

```
-Xcpu=core
```

- Interpretation when omitted
  If the -Xcommon option is specified, 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.
  An abort error will occur if any other item is specified.

g3m	Generates an object for G3M.
g3k	Generates an object for G3K.

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

[Example of use]

- To generate an object for G3M, describe as:

```
>asrh -Xcpu=g3m -c -Xcommon=v850e3v5 main.asm
```

Preprocessor control

The preprocessor control options are as follows.

- -D
- -U
- -I

-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.
- This is equivalent to adding "*name* .SET def" at the beginning of the assembly 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.
- 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=v850e3v5 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=v850e3v5 main.asm
```

-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

(2) Folder with source file

(3) Folder of original C source file

(4) 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=v850e3v5 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=v850e3v5 main.asm
```

Generated code control

The generated code control options are as follows.

- -Xreg_mode
- -Xreserve_r2
- -Xep

-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.
- An abort error will occur if the register that cannot be used for the assembly source file.
- The same register mode must be specified for all source files.
  The different register mode cannot be specified for each source file.
  If there are object files with different register modes, an error will occur during linking.
- By specifying this option, the register mode of the software register bank function can be changed.

[Example of use]

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

```
>asrh -Xreg_mode=22 -Xcommon=v850e3v5 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=v850e3v5 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 any of the following attribute strings in a #pragma section in the project is used. ep_auto, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23
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.

[Example of use]

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

```
>asrh -Xep=fix -Xcommon=v850e3v5 main.asm
```

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=v850e3v5 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=v850e3v5 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_waning=num[,num]...
-Xno_waning=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.

[Example of use]

- To suppress outputting warning message "W0550002" and "W0550003", describe as:

```
>asrh -Xno_waning=50002,50003 -Xcommon=v850e3v5 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=v850e3v5
```

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.
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.
	-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.
	-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.
List output	-LISt	This option outputs the list file.
	-SHow	This option specifies information that is output to the list file.

Classification	Option	Description
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.
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.
	-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.
	-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

-Input

This option specifies the input file.

[Specification format]

```
-Input=file[(module[,module]...)][{,|Δ}file[(module[,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=file(section[:alignment][/attribute][,symbol])[,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=symbol1={symbol2|value}[,symbol1={symbol2|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.

Output control

The output control options are as follows.

- -FOrm
- -DEBug
- -NODEBug
- -RECord
- -ROm
- -OUtput
- -MAp
- -SPace
- -Message
- -NOMessage
- -MSg_unused
- -BYte_count
- -PADDING
- -OVERRUN_FETCH

-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.

- If *format* is omitted, it is assumed that "Absolute" has been specified.

[Remark]

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

Figure 2.2 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, -hide, -compress, -rename, -delete, -define, -fsymbol, -stack, -memory, -msg_unused, -show={symbol|reference|xreference}, -aligned_section, -overrun_fetch

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, -debug, -nodebug, -output, -hide, -rename, -delete, -show={symbol|xreference}
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, -rename, -delete, -define, -fsymbol, -stack, -record, -s9Note 2, -byte_countNote 3, -memory, -msg_unused, -show={symbol|reference|xreference}, -aligned_section, -overrun_fetchNote 4
		Load module file	-input, -output, -record, -s9Note 2, -byte_countNote 3, -show={symbol|reference|xreference}
		Intel HEX fileNote 5	-input, -output
		Motorola S-record fileNote 5	-input, -output, -s9Note 2
Library	-strip specified	Library file	-library, -output, -memoryNote 6, -show={symbol|section}
	-extract specified	Library file	-library, -output
	Other than above	Object file Relocatable file	-input, -library, -output, -hide, -rename, -delete, -replace, -memoryNote 6, -show={symbol|section}

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

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

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

[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.

-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 a nonexistent section or relocatable section whose size is 0 for RAM section *RAMsection*.

[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=file[={address1-address2|section[:section]...}][,file[={address1-
address2|section[:section]...}]]...
```

- Interpretation when omitted
  The output file name is "*first-input-file-name.default-extension*".
  The default extensions are shown below.
  - When the -form=absolute option is specified: abs
  - When the -form=relocate option is specified: rel
  - When the -form=object option is specified: obj
  - When the -form=library option is specified: lib
  - When the -form=hexadecimal option is specified: hex
  - When the -form=stype option is specified: mot
  - When the -form=binary option is specified: bin

[Detailed description]

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

[Example of use]

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

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

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

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

[Remark]

- If a input file is an Intel Hex file or Motorola S-record file, two or more output files cannot be specified by this option.
  If this option is omitted, the output file name will be "*first input file name*_combine.*extension*" (If the input file is "a.mot", the output file will be "a_combine.mot").

-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.

[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 following vacant areas are filled with the value according to the output range specification in the -output option:
  - When the section is specified as the output range
    - The specified value is output to vacant areas between the specified sections.
  - When the start address and end address is specified as the output range
    - The specified value is output to vacant areas within the specified range.
- 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, 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[-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
  An Intel HEX file is generated assuming that the maximum byte count is "FF".

[Detailed description]

- This option specifies maximum byte count for a data record when a file *num* when an Intel HEX file.
- Specify a one-byte hexadecimal value (01 to FF) as *num*.

[Example of use]

- To specify 10 as the maximum byte count for a data record, describe as:

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

[Remark]

- When a generated file is not an Intel HEX file (when the -form=hexadecimal 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 P section, and linking is performed with a size of 0x08.
  Alignment of the P section: 4 bytes
  Size of the P section: 0x06 bytes
  Alignment of the C section: 1 byte
  Size of the C section: 0x03 bytes

```
>rlink a.obj b.obj -start=P,C/0 -padding
```

- In the following case, if 2 bytes of padding data are filled in the P section, and linking is performed with a size of 0x08, then an error will be output because it overlaps with the C section.
  Alignment of the P section: 4 bytes
  Size of the P section: 0x06 bytes
  Alignment of the C section: 1 byte
  Size of the C section: 0x03 bytes

```
>rlink a.obj b.obj -start=P/0,C/7 -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 vacant areas due to overrun fetch.

[Specification format]

```
-OVERRUN_FETCH
```

- Interpretation when omitted
  None

[Detailed description]

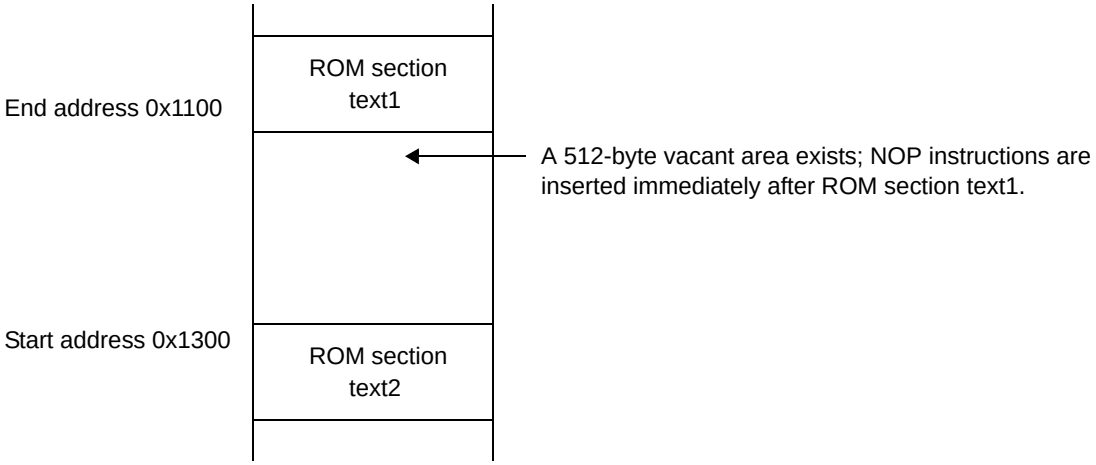
- 128-byte NOP instructions are inserted in a vacant area when a 128-byte or larger vacant area exists between sections or immediately after the end code section.
  NOP instructions are inserted according to the allocation specified by the -start option and output with the following section name obtained 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

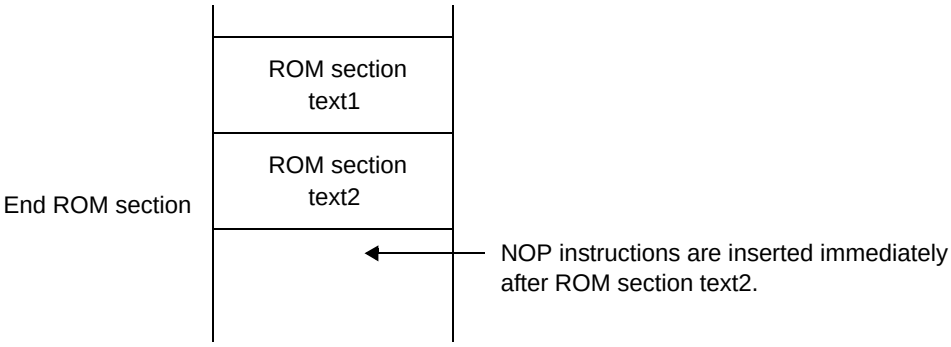
- NOP instructions are inserted when a 128-byte or larger vacant area exists between two ROM sections (executable program sections or sections (except program sections) allocated to the ROM area) or between a ROM section and a RAM section (a section allocated to the RAM area).



- 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:

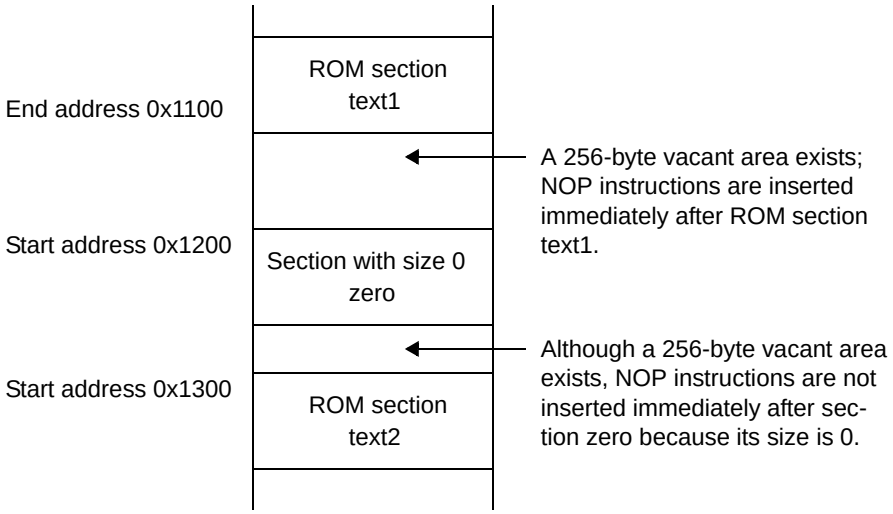
```
>rlink a.obj b.obj -overrun_fetch
```

[Remark]

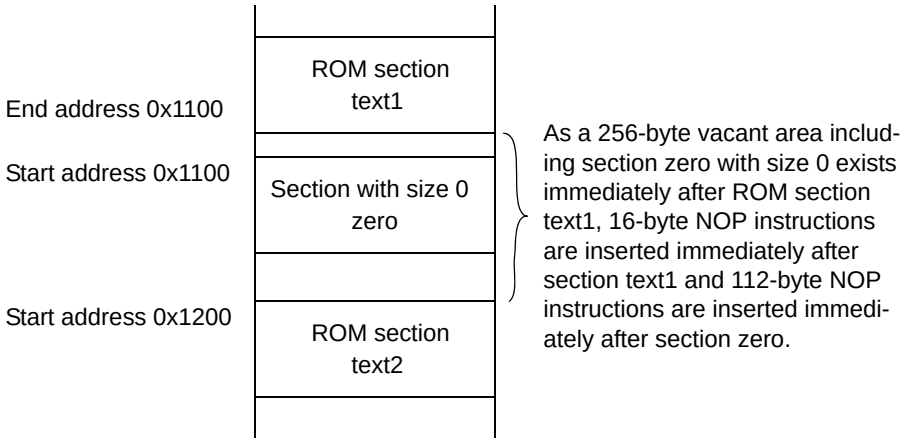
- When no address range is specified by the -cpu option and the end section is a ROM section, NOP instructions are always inserted immediately after the end ROM section.
- If a load module file is input, this option will be invalid.
- If the -form={object|relocate|library} option is specified, this option 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.


ROM section
text1
End address 0x1100
Start address 0x1100
Section with size 0
zero
Start address 0x1200
ROM section
text2
As a 256-byte vacant area including section zero with size 0 exists immediately after ROM section text1, 16-byte NOP instructions are inserted immediately after section text1 and 112-byte NOP instructions are inserted immediately after section zero.

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.

[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).
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 option is specified. When the -form=hexadecimal/stype/binary option Outputs the same information as when the -show=symbol,reference,xreference,total_size option is specified. When the -form=object option is specified Not specifiable

Remark See "3.2 Link Map File" and "3.4 Library List File" for details about output information.

- See [Remark] for details about when the specification of output information is omitted.

[Example of use]

- To output the symbol address, size, type, optimization contents, and number of symbol references, describe as:

```
>rlink a.obj b.obj -list -show=symbol,reference
```

[Remark]

- The following table shows whether output information *info* will be valid or invalid by the combinations of the -form option and the -show or -show=all option.

		Symbol	Reference	Section	Xreference	Total_size
-form=absolute	only -show	Valid	Valid	Invalid	Invalid	Invalid
	-show=all	Valid	Valid	Invalid	Valid	Valid
-form=library	only -show	Valid	Invalid	Valid	Invalid	Invalid
	-show=all	Valid	Invalid	Valid	Invalid	Invalid
-form=relocate	only -show	Valid	Invalid	Invalid	Invalid	Invalid
	-show=all	Valid	Invalid	Invalid	Valid[Note 1]	Valid
-form=object	only -show	Valid	Valid	Invalid	Invalid	Invalid
	-show=all	Invalid	Invalid	Invalid	Invalid	Invalid
-form=hexadecimal[Note 2]/ stype[Note 3]/binary	only -show	Valid	Valid	Invalid	Invalid	Invalid
	-show=all	Valid	Valid	Invalid	Valid	Valid[Note 1]

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.

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.

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=[(]section[{:|,}section]...[)][{:|,}section]...[/address][,[(]section[{:|,}
section]...[)][{:|,}section]...[/address]]...
```

- 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.

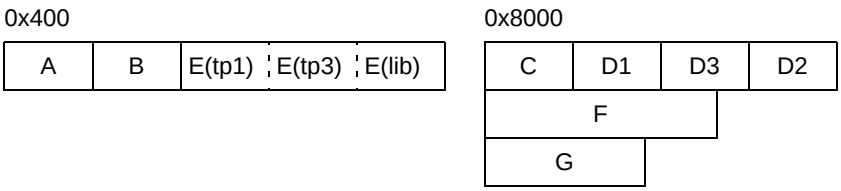
[Detailed description]

- 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)

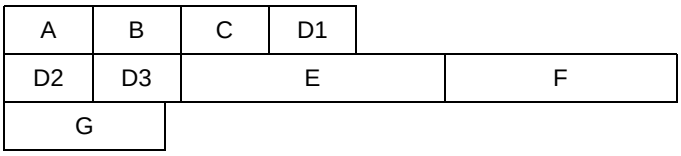
  - When the -start=A,B,E/400,C,D*:F:G/8000 option is specified



  - 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

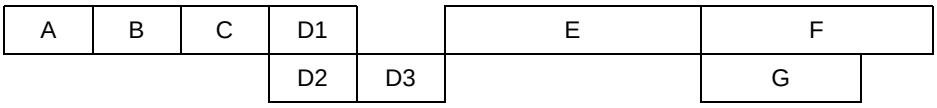
0x400



  - 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 "()".

-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=type=address1-address2[,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
- -DELete
- -REPlace
- -EXTract
- -STRip
- -CHange_message
- -Hide
- -Total_size
- -LOgo
- -NOLOgo
- -END
- -EXIt

-S9

This option outputs the S9 record at the end.

[Specification format]

```
-S9
```

- Interpretation when omitted
  None

[Detailed description]

- This option outputs the S9 record at the end even if the entry address exceeds 0x10000.

[Example of use]

- To output the S9 record at the end even if the entry address 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 -compress, -delete, -rename, -map, or -stack options
    - The -list and -show[={reference|xreference}] options are specified at the same time.
  - When the -form=library option and following options are specified at the same time
    - Any of the -delete, -rename, -extract, -hide, or -replace 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 "Figure 2.2 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={[file](name1=name2[,name1=name2]...)|[module](name1=name2[,name1=name2]...)}
[[,{[file](name1=name2[,name1=name2]...)|[module](name1=name2[,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*.
- Symbol names or section names in specific file *file* or a module in specific library *module* can be changed.
- When a C variable name is specified, add "_" at the head of the definition name in the program.
- When a function name is changed, the operation is not guaranteed.
- 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.

[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]

- If this option is specified together with the -extract option or -strip option, this option will be invalid.
- When the -form=absolute 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.

-DELete

This option deletes an external symbol name or a library module.

[Specification format]

```
-DELete={[file](symbol[,symbol]...)|module}[,{[file](symbol[,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=file[(module[,module]...)][,file[(module[,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=level[=num[-num][,num[-num]]...][,level[=num[-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.
- 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-support function (-rom option) has been specified for a section, the section will be used by both the source (ROM) and destination (RAM) of the transfer. The sizes of sections of this type will be added to the total sizes of sections in both ROM and RAM.

-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.

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 -O*item*: -Ounroll, -Oinline, -Oinline_size, -Odelete_static_func, -Opipeline, -Otail_call, -Omap, -Osmap

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, sdata, sdata23, ep_auto, ep_disp4, ep_disp5, ep_disp7, ep_disp8, ep_disp16, ep_disp23, gp_disp16, gp_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=v850e3v5 -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=v850e3v5 -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
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	No output
Cross reference information	Symbol reference information	-show=xreference	No output

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
```

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.

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.

Number	Description
(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
(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 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.
(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*)

3.3 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.

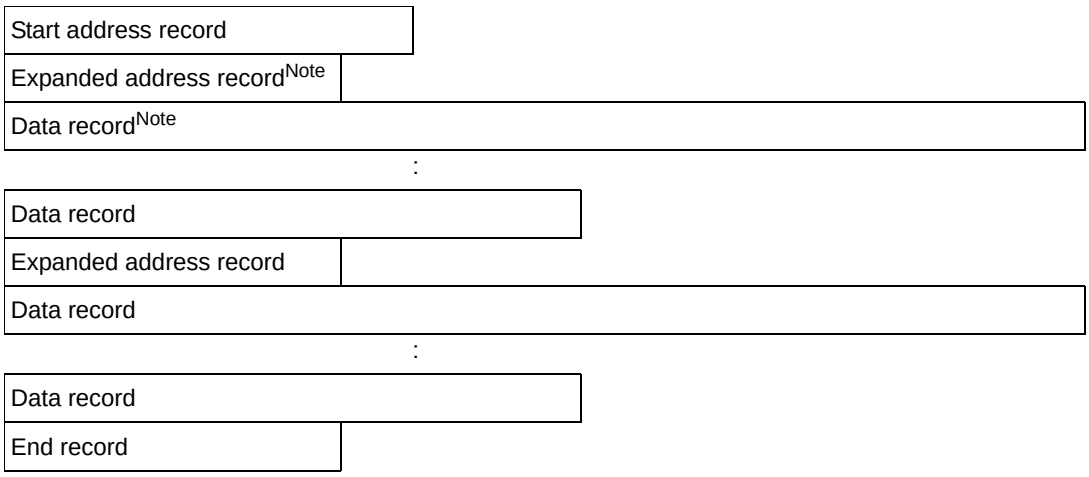
3.5.1 Structure of the Intel HEX file

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

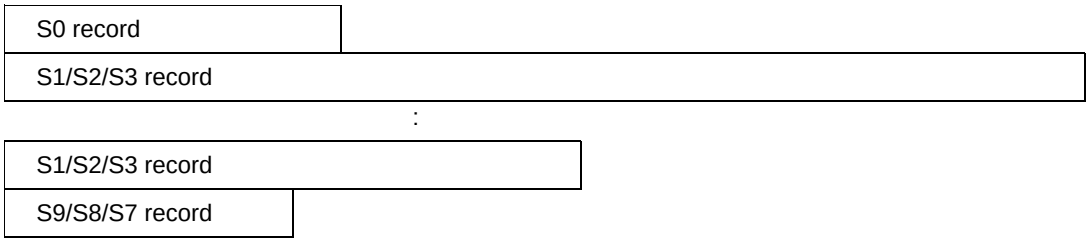
The Motorola S-record file consists of seven records[Note 1]: S0 record as the header record, S1, S2, and S3 records as the data record, and S9, S8, and S7 records as the end records[Note 2].

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

The CC-RH supports the language specifications stipulated by the ANSI standards. These specifications include items that are stipulated as processing definitions. This section describes language specifications that are not specified or defined by the ANSI standard, or that are process-dependent.

It describes the differences between when the option for processing in strict compliance with the ANSI standard (-Xansi) is specified, and when it is not.

See "4.2 Extended Language Specifications" for extended language specifications explicitly added by the CC-RH.

4.1.1 Unspecified behavior

This section describes behavior that is not specified by the ANSI standard.

(1) Execution environment - initialization of static storage
Static data is output during compilation as a data section.

(2) Meanings of character displays - final line location, backspace (\b), horizontal tab (\t), vertical tab (\t)
This is dependent on the design of the display device.

(3) Types - floating point
IConforms to IEEE754[Note].

Note IEEE: Institute of Electrical and Electronics Engineers
IEEE754 is a system for handling floating-point calculations, providing a uniform standard for data formats, numerical ranges, and the like handled.

(4) Expressions - evaluation order
Unspecified.

(5) Function calls - parameter evaluation order
Unspecified.

(6) Structure and union specifiers
See "(8) Bit field" for the boundary alignment of structure objects with bit fields.

(7) Function definitions - storage of formal parameters
Unspecified.

(8) # operator
Unspecified.

4.1.2 Undefined behavior

This section describes behavior that is not defined by the ANSI standard.

(1) Character set
A message is output if a source file contains a character not specified by the character set.

(2) Lexical elements
A message is output if there is a single or double quotation mark ("/") in the last category (a delimiter or a single non-whitespace character that does not lexically match another preprocessing lexical type).

(3) Identifiers
Since all identifier characters have meaning, there are no meaningless characters.

(4) Identifier binding
A message is output if both internal and external binding was performed on the same identifier within a translation unit.

(5) Compatible type and composite type
All declarations referencing the same object or function must be compatible. Otherwise, a message will be output.

(6) Character constants

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.

(7) String literals - concatenation
When a simple string literal is adjacent to a wide string literal token, they are concatenated into a wide string literal.

(8) String literals - modification
Users modify string literals at their own risk.

(9) Header names
If the following characters appear in strings between the delimiter characters < and >, or between two double quotation marks ("), then they are treated as part of the file name: characters, comma (,), double quote ("), two slashes (//), or slash-asterisk (/*). The backslash (\) is treated as a folder separator.

(10) Floating point type and integral type
If a floating-point type is typecast into an integral type, and the integer portion cannot be expressed as an integral type, then it is undefined.

(11) lvalues and function specifiers
If an incomplete type becomes an value, then it is undefined..

(12) Function calls - number of arguments
If there are not enough actual parameters, then it is undefined.

(13) Function calls - types of extended parameters
If a function is defined without a function prototype, and the types of the extended arguments do not match the types of the extended formal parameters, then the values of the formal parameters will be undefined.

(14) Function calls - incompatible types
If a function is defined with a type that is not compatible with the type specified by the expression indicating the called function, then the return value of the function will be invalid.

(15) Function calls - incompatible types
If a function is defined in a form that includes a function prototype, and the type of an extended argument is not compatible with that of a formal parameter, or if the function prototype ends with an ellipsis, then it will be interpreted as the type of the formal parameter.

(16) Addresses and indirection operators
If an incorrect value is assigned to a pointer, then the behavior of the unary * operator will either obtain an undefined value or result in an illegal access, depending on the hardware design and the contents of the incorrect value.

(17) Cast operator - function pointer casts
If a typecast pointer is used to call a function with other than the original type, then it is undefined.

(18) Cast operator - integral type casts
If a pointer is cast into an integral type, and the amount of storage is too small, then it is undefined.

(19) Multiplicative operators
A message will be output if a divide by zero is detected during compilation.

(20) Additive operators - non-array pointers
If addition or subtraction is performed on a pointer that does other than indicate elements in an array object, then it is undefined.

(21) Additive operators - subtracting a pointer from another array
If subtraction is performed using two pointers that do not indicate elements in the same array object, then it is undefined.

(22) Bitwise shift operators
If the right operand is negative or the expanded left operand is wider than the bit width, then it is undefined.

(23) Function operators - pointers
If the objects referred to by the pointers being compared are not members of the same aggregate or union object, then it is undefined.

(24) Simple assignment
If a value stored in an object is accessed via another object that overlaps that object's storage area in some way, then the overlapping portion must match exactly. Furthermore, the types of the two objects must have modified or non-modified versions with compatible types. Assignment to non-matching overlapping storage could cause the value of the assignment source to become corrupted.

(25) Structure and union specifiers
If the member declaration list does not include named members, then a message will be output warning that the list has no effect. Note, however, that the same message will be output accompanied by an error if the -Xansi option is specified.

(26) Type modifiers - const
If an object defined with a const modifier is modified using an lvalue that is the non-const modified version, then it is undefined.

(27) Type modifiers - volatile
If an object defined with a const modifier is modified using an lvalue that is the non-const modified version, then it is undefined.

(28) return statements
A message will be output if a return statement without an expression is executed, and the caller uses the return value of the function, and there is a declaration. If there is no declaration, then the return value of the function will be undefined.

(29) Function definitions
If a function taking a variable number of arguments is defined without a parameter type list that ends with an ellipsis, then the values of the formal parameters will be undefined.

(30) Conditional inclusion
If a replacement operation generates a "defined" token, or if the usage of the "defined" unary operator before macro replacement does not match one of the two formats specified in the constraints, then it will be handled as an ordinary "defined".

(31) Macro replacement - arguments not containing preprocessing tokens
A message is output if the arguments (before argument replacement) do not contain preprocessing tokens.

(32) Macro replacement - arguments with preprocessing directives
A message is output if an argument list contains a preprocessor token stream that would function as a processing directive in another circumstance.

(33) # operator
A message is output if the results of replacement are not a correct simple string literal.

(34) ## operator
A message is output if the results of replacement are not a correct simple string literal.

4.1.3 Processing system dependent items

This section explains items dependent on processing system in the ANSI standards.

(1) Data types and sizes
The byte order in a word (4 bytes) is "from least significant to most significant byte" Signed integers are expressed by 2's complements. The sign is added to the most significant bit (0 for positive or 0, and 1 for negative).

- The number of bits of 1 byte is 8.
- The number of bytes, byte order, and encoding in an object files are stipulated below.

Table 4.1 Data Types and Sizes

Data Types	Sizes
char	1 byte
short	2 bytes
int, long, float	4 bytes
double, long double, long long	8 bytes
pointer	Same as unsigned int

(2) Translation stages
The ANSI standards specify eight translation stages (known as "phases") of priorities among syntax rules for translation. The arrangement of "non-empty white space characters excluding line feed characters" which is

defined as processing system dependent in phase 3 "Decomposition of source file into preprocessing tokens and white space characters" is maintained as it is without being replaced by single white space character.

(3) Diagnostic messages
When syntax rule violation or restriction violation occurs on a translation unit, the compiler outputs as error message containing source file name and (when it can be determined) the number of line containing the error. These error messages are classified: "Warning", "Abort error", "Fatal error" and "other" messages. For output formats of messages, see the "CS+ Integrated Development Environment User's Manual: Message".

(4) Program startup processing
The name and type of a function that is called on starting program processing are not stipulted.
Therefore, it is dependent on the user-own coding and target system.

(5) Program execution
The configuration of the interactive unit is not stipulated.
Therefore, it is dependent on the user-own coding and target system.

(6) Character set
The values of elements of the execution environment character set are ASCII codes.

(7) Multi-byte characters
Supported multi-byte characters are ECU, SJIS, UTF-8, big5 and gb2312.
Japanese and Chinese description in comments and character strings is supported.

(8) Significance of character display
The values of expanded notation are stipulated as follows.

Table 4.2 Expanded Notation and Meaning

Expanded Notation	Value (ASCII)	Meaning
\a	07	Alert (Warning tone)
\b	08	Backspace
\f	0C	Form feed (New Page)
\n	0A	New line (Line feed)
\r	0D	Carriage return (Restore)
\t	09	Horizontal tab
\v	0B	Vertical tab

(9) Translation limit
There are no limits on translation. The maximum translatable value depends on the memory of the host machine on which the program is running.

(10) Quantitative limit

(a) The limit values of the general integer types (limits.h file)
The limits.h file specifies the limit values of the values that can be expressed as general integer types (char type, signed/unsigned integer type, and enumerate type).
Because multi-byte characters are not supported, MB_LEN_MAX does not have a corresponding limit. Consequently, it is only defined with MB_LEN_MAX as 1.
The limit values defined by the limits.h file are as follows.

Table 4.3 Limit Values of General Integer Type (limits.h File)

Name	Value	Meaning
CHAR_BIT	+8	The number of bits (= 1 byte) of the minimum object not in bit field
SCHAR_MIN	-128	Minimum value of signed char
SCHAR_MAX	+127	Maximum value of signed char
UCHAR_MAX	+255	Maximum value of unsigned char

Name	Value	Meaning
CHAR_MIN	-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

(b) The limit values of the floating-point type (float.h file)
The limit values related to characteristics of the floating-point type are defined in float.h file.
The limit values defined by the float.h file are as follows.

Table 4.4 Definition of Limit Values of Floating-point Type (float.h File)

Name	Value	Meaning
FLT_ROUNDS	+1	Rounding mode for floating-point addition. 1 for the RH850 family (rounding in the nearest direction).
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	
LDBL_MANT_DIG	+53	
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	
LDBL_DIG	+15	
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	
LDBL_MIN_EXP	-1021	
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	
LDBL_MIN_10_EXP	-307	

Name	Value	Meaning
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	
LDBL_MAX_EXP	+1024	
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	
LDBL_MAX_10_EXP	+308	
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	
LDBL_MAX	1.7976931348623158E+308	
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	
LDBL_EPSILON	2.2204460492503131E-016	
FLT_MIN	1.17549435E - 38F	Minimum value of normalized positive floating-point number $b^{e_{min} - 1}$
DBL_MIN	2.2250738585072014E-308	
LDBL_MIN	2.2250738585072014E-308	

(11) Identifier
All identifiers are considered to have meaning. There are no restrictions on identifier length.
Uppercase and lowercase characters are distinguished.

(12) char type
If a value larger than a char is stored in a char type, then the value is converted to type char.
A char type with no type specifier (signed, unsigned) specified is treated as a signed integer as the default assumption.

```
char    c = '\777';     /*Value of c is -1.*/
```

(13) Floating-point constants
The floating-point constants conform to IEEE754[Note].

Note IEEE: Institute of Electrical and Electronics Engineers
IEEE754 is a system for handling floating-point calculations, providing a uniform standard for data formats, numerical ranges, and the like handled.

(14) Character constants

(a) Both the character set of the source program and the character set in the execution environment are basically ASCII codes, and correspond to members having the same value.
However, for the character set of the source program, character codes in Japanese can be used (see "(8) Significance of character display").

(b) The last character of the value of an integer character constant including two or more characters is valid.

(c) A character that cannot be expressed by the basic execution environment character set or escape sequence is expressed as follows.

<1> An octal or hexadecimal escape sequence takes the value indicated by the octal or hexadecimal notation

\777	511

<2> The simple escape sequence is expressed as follows.

\'	'

\"	"
\?	?
\\	\

<3> Values of \a, \b, \f, \n, \r, \t, \v are same as the values explained in "(8) Significance of character display".

(d) Character constants of multi byte characters are not supported.

(15) Character string
A character string can be described in Japanese and Chinese.
The default character code is Shift JIS.
A character code in input source file can be selected by using the -Xcharacter_set option of the CC-RH.

```
-Xcharacter_set=[none | euc_jp | sjis | utf8 | big5 | gb2312]
```

(16) Header file name
The method to reflect the string in the two formats (< > and " ") of a header file name on the header file or an external source file name is stipulated in "(33) Loading header file".

(17) Comment
A comment can be described in Japanese and Chinese. The character code is the same as the character string in "(15) Character string".

(18) Signed constants and unsigned constants
If the value of a general integer type is converted into a signed integer of a smaller size, the higher bits are truncated and a bit string image is copied.
If an unsigned integer is converted into the corresponding signed integer, the internal representation is not changed.

(19) Floating-points and general integers
If the value of a general integer type is converted into the value of a floating-point type, and if the value to be converted is within a range that can be expressed but not accurately, the result is rounded to the closest expressible value.

(20) double type and float type
When casting a double to a float, or a long double to a double or a float, if the typecast value cannot be represented accurately in the available value range, then the result will be rounded to the nearest value that can be represented.

(21) Signed type in operator in bit units
The characteristics of the shift operator conform to the stipulation in "(27) Shift operator in bit units".
The other operators in bit units for signed type are calculated as unsigned values (as in the bit image).

(22) Members of structures and unions
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, access to members of the union will be of the subsequently accessed type.

(23) sizeof operator
The value resulting from the "sizeof" operator conforms to the stipulation related to the bytes in an object in "(1) Data types and sizes".
For the number of bytes in a structure and union, it is byte including padding area.

(24) Cast operator
When a pointer is converted into a general integer type, the required size of the variable is the same as the size of the unsigned long type. The bit string is saved as is as the conversion result.
Also, although it is possible to typecast an arbitrary integer to a pointer, after the integer is typecast to a pointer, the integer's bit pattern is retained unchanged.

(25) Division/remainder operator
The result of the division operator ("/") when the operands are negative and do not divide perfectly with integer division, is as follows: If either the divisor or the dividend is negative, the result is the smallest integer greater than the algebraic quotient.
If both the divisor and the dividend are negative, the result is the largest integer less than the algebraic quotient.
If the operand is negative, the result of the "%" operator takes the sign of the first operand in the expression.

(26) Addition and subtraction operators

If two pointers indicating the elements of the same array are subtracted, the type of the result is unsigned long type.

(27) Shift operator in bit units
If E1 of "E1 >> E2" is of signed type and takes a negative value, an arithmetic shift is executed.

(28) Storage area class specifier
Optimize for the fastest possible access, regardless of whether there is a storage-class area specifier "register" declaration.

(29) Structure and union specifier

(a) A simple int type bit field without signed or unsigned appended is treated as a signed field, and the most significant bit is treated as the sign bit.

(b) To retain a bit field, a storage area unit to which any address with sufficient size can be assigned can be allocated. If there is insufficient area, however, the bit field that does not match is packed into to the next unit according to the alignment condition of the type of the field.
The allocation sequence of the bit field in unit is from lower to higher.

(c) Each member of the non-bit field of one structure or union is aligned at a boundary as follows:

char, unsigned char type, and its array	Byte boundary
short, unsigned short type, and its array	2-byte boundary
Others (including pointer)	4-byte boundary

If packing is used, however, the boundaries of all members will be the byte boundary.

(30) Enumerate type specifier
The type of an enumeration specifier is signed int.
However, when the -Xenum_type=auto option is specified, each enumerated type is treated as the smallest integer type capable of expressing all the enumerators in that type.

(31) Type qualifier
The configuration of access to data having a type qualified to be "volatile" is dependent upon the address (I/O port, etc.) to which the data is mapped. If packing is used, however, the boundaries of all members will be the byte boundary.

(32) Condition embedding

(a) The value for the constant specified for condition embedding and the value of the character constant appearing in the other expressions are equal.

(b) A single-character character constant cannot have a negative value.

(33) Loading header file

(a) A preprocessing directive in the form of "#include <character string>"
A preprocessing directive in the form of "#include <character string>" searches for a header file from the folder specified by the -I option if "character string" does not begin with "\"[Note], and then searches standard include file folder (..\inc folder with a relative path from the bin folder where the ccrh is placed).
If a header file uniformly identified is searched with a character string specified between delimiters "<" and ">", the whole contents of the header file are replaced.

Note "/" are regarded as the delimiters of a folder.

Example

```
#include    <header.h>
```

The search order is as follows.

- Folder specified by -I
- Standard include file folder

(b) A preprocessing directive in the form of "#include "character string""

A preprocessing directive in the form of "#include "character string"" searches for a header file from the folder where the source file exists, then searches specified folder (-I option) and then searches standard include file folder (..\inc folder with a relative path from the bin folder where the ccrh is placed).
If a header file uniformly identified is searched with a character string specified between delimiters " " " and " " ", the whole contents of the header file are replaced.

Example

```
#include    "header.h"
```

The search order is as follows.

- Folder where source file exists
- Folder specified by -I
- Standard include file folder

(c) The format of "#include preprocessing character phrase string"
The format of "#include preprocessing character phrase string" is treated as the preprocessing character phrase of single header file only if the preprocessing character phrase string is a macro that is replaced to the form of <character string> or "character string".

(d) A preprocessing directive in the form of "#include <character string>"
Between a string delimited (finally) and a header file name, the length of the alphabetic characters in the strings is identified,

And the file name length valid in the compiler operating environment is valid.

The folder that searches a file conforms to the above stipulation.

(34) #pragma directive
The CC-RH can specify the following #pragma directives.

(a) Data or program memory allocation

```
#pragma section section-type ["section-name"]
```

Allocates variables to an arbitrary section.
For details about the allocation method, see "4.2.5.1 Allocation of data and program to section".

(b) Describing assembler instruction

```
#pragma inline_asm
```

Assembler directives can be described in a C source program.
For the details of description, see "4.2.5.2 Describing assembler instruction".

(c) Inline expansion specification

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

A function that is expanded inline can be specified.
For the details of expansion specification, see "4.2.5.3 Inline expansion".

(d) Interrupt/exception handler specification

```
#pragma interrupt interrupt-request-name function-name [allocation-method]
[Option [Option]...]
```

Interrupt/Exception handlers are described in C language.
For the details of description, see "(3) Describing interrupt/exception handler".

(e) Interrupt disable function specification

```
#pragma block_interrupt [(]function-name[)]
```

Interrupts are disabled for the entire function.
For details of description, see "(2) Disabling interrupts in entire function".

(f) Structure type packing specification

```
#pragma pack [(][1|2|4][)]
```

Specifies the packing of a structure type. The packing value, which is an alignment value of the member, is specified as the numeric value. A value of 1, 2, 4 can be specified. When the numeric value is not specified, it is by default alignment.
For details of description, see "4.2.5.8 Structure type packing".

(g) Specifying bit field assignment

```
#pragma bit_order [{left|right}]
```

This specifies a change to the order of the bit field.
For details of description, see "4.2.5.9 Bit field assignment".

(h) Core number specification (for a multi-core device)

```
#pragma pmodule pm-specification
```

A function to which a core number is to be assigned can be specified.
For details of description, see "4.2.5.10 Core number specification (for a multi-core device)".

(35) Predefined macro names
All the following macro names are supported.

Table 4.5 List of Supported Macros

Macro Name	Definition
__LINE__	Line number of source line at that point (decimal).
__FILE__	Name of source file (character string constant).
__DATE__	Date of translating source file (character string constant in the form of "Mmm dd yyyy"). Here, the name of the month is the same as that created by the asctime function stipulated by ANSI standards (3 alphabetic characters with only the first character is capital letter) (The first character of dd is blank if its value is less than 10).
__TIME__	Translation time of source file (character string constant having format "hh:mm:ss" similar to the time created by the asctime function).
__STDC__	Decimal constant 1 (defined when the -Xansi option is specified).[Note]
__RENESAS__	This value is not set.
__RENESAS_VERSION__	If the version is V.XX.YY.ZZ, this will be 0xXXYYZZ00. Example) V.1.00.00 -> -D__RENESAS_VERSION__=0x01000000
__CCRH__	This value is not set .
__CCRH	This value is not set.
__RH850__	This value is not set.
__RH850	This value is not set.
__v850e3v5__	This value is not set (defined when v850e3v5 or rh850 is specified by the -Xcommon option is specified).
__v850e3v5	This value is not set (defined when v850e3v5 or rh850 is specified by the -Xcommon option is specified).
__DBL8	Supported when -Xdbl_size=8 is specified or -Xdbl_size is not specified. This value is not set.

Macro Name	Definition
__DBL4	Supported only when -Xdbl_size=4 is specified. This value is not set.
__DOUBLE_IS_64BITS__	Supported when -Xdbl_size=8 is specified or -Xdbl_size is not specified. This value is not set.
__DOUBLE_IS_32BITS__	Supported only when -Xdbl_size=4 is specified. This value is not set.
__RON	Supported only when -Xround=nearest is specified. This value is not set.
__ROZ	Supported only when -Xround=zero is specified. This value is not set.
__BITLEFT	This value is not set (defined when left is specified by the -Xbit_order option).
__BITRIGHT	This value is not set (defined when right is specified by the -Xbit_order option).
__AUTO_ENUM	This value is not set (defined when auto is specified by the -Xenum_type option).
__FPU	This value is not set (defined when fpu is specified by the -Xfloat option).
__CHAR_SIGNED__	This value is not set..
Register mode macro	Since this macro indicates the target CPU, its value is not set. Macro defined with register mode is as follows. 32 register mode: __reg32__ 22 register mode: __reg22__ Universal register mode: __reg_common__
_LIT	This value is not set.
__MULTI_LEVEL__	level value designated in Xmulti_level = level

Note For the processing to be performed when the -Xansi option is specified, see "4.1.5 Option to process in strict accordance with ANSI standard".

4.1.4 C99 language function

This section describes the C99 language functions supported by the CC-RH.

(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.

(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, ...][)]
```

For the details of expansion specification, see "4.2.5.3 Inline expansion".

(9) Types of integer constants
In accordance with the addition of type long long, the types of integer constants are as follows.
The type of an integer constant will be the first type in the lists below capable of representing that value.

Table 4.6 Types of Integer Constants (If type long long is enabled (when -Xansi is not specified))

Suffix	Decimal Constant	Octal Constant or Hexadecimal
None	int long int unsigned long int[Note] 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	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	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 4.7 Types of Integer Constants (If type long long is disabled (when -Xansi is specified))

Suffix	Decimal Constant	Octal Constant or Hexadecimal
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	long int unsigned long int
Both u or U, and l or L	unsigned long int	unsigned long int

4.1.5 Option to process in strict accordance with ANSI standard

If the -Xansi option is specified to CC-RH, it will perform processing in strict accordance with the ANSI standard. The differences between when the -Xansi option is specified and when not specified are as follows.

Table 4.8 Processing When -Xansi Option Strictly Conforming to Language Specifications is Specified

Item	With -Xansi Specification	Without -Xansi Specification
Bit field	An error[Note 1] occurs if type other than int is specified for bit field.	Specification of types other than int is allowed.
# line number	An error occurs.	Treated in same manner as "#line line number".[Note 2]
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.	The type of the return value is converted to the type at the call site, the parameters are converted to the type of the function definition, and inline expansion is performed.
__STDC__	Defines value as macro with value 1.	Does not define.
_Bool type long long type	An error occurs.	Specification is allowed.
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.	If the member declaration list does not include named members, then an warning message will be output indicating that the list has no effect.

Note 1. Normal error beginning with "E". The same applies hereafter.

Note 2. See the ANSI standards.

4.1.6 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.

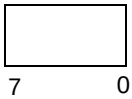
(1) Integer type

(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.

Figure 4.1 Internal Representation of Integer Type

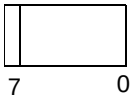
_Bool



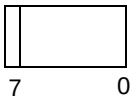
Only the 0th bit has meaning. Bits 1 to 7 are undefined.

When the -Xansi option is used, _Bool type will cause a C90 violation error.

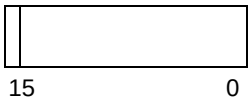
char



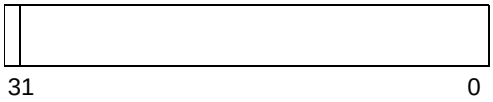
signed char (no sign bit for unsigned)



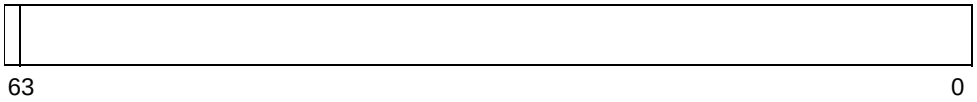
short (no sign bit for unsigned)



int, long (no sign bit for unsigned)



long long (no sign bit for unsigned)



When the -Xansi option is used, long long type will cause a C90 violation error.

(b) Value area

Table 4.9 Value Area of Integer Type

Type	Value Area
char[Note]	-128 to +127
short	-32768 to +32767
int	-2147483648 to +2147483647

Type	Value Area
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

(2) Floating-point type

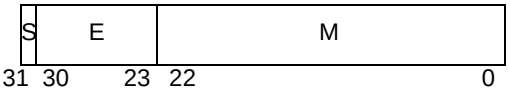
(a) Internal representation

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.

Figure 4.2 Internal Representation of Floating-Point Type

float

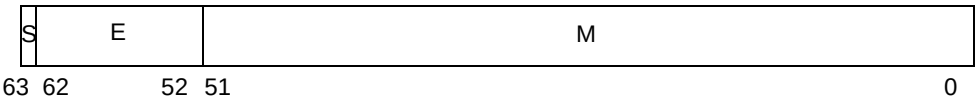


S: Sign bit of mantissa

E: Exponent (8 bits)

M: Mantissa (23 bits)

double



S: Sign bit of mantissa

E: Exponent (11 bits)

M: Mantissa (52 bits)

(b) Value area

Table 4.10 Value Area of Floating-Point Type

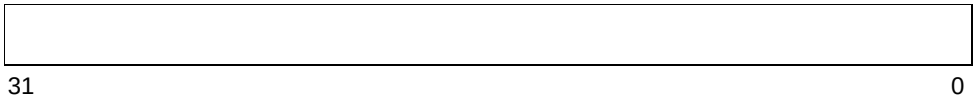
Type	Value Area
float	1.17549435E-38F to 3.40282347E+38F
double	2.2250738585072014E-308 to 1.7976931348623158E+308
long double	2.2250738585072014E-308 to 1.7976931348623158E+308

(3) Pointer type

(a) Internal representation

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

Figure 4.3 Internal Representation of Pointer Type

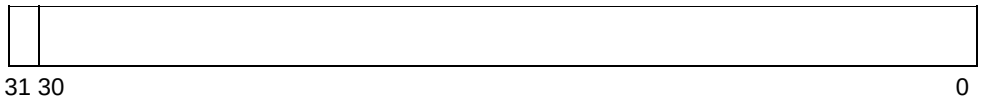


(4) Enumerate type

(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 4.4 Internal Representation of Enumerate Type



When the -Xenum_type=auto option is specified, see "(30) Enumerate type specifier".

(5) Array type

(a) Internal representation

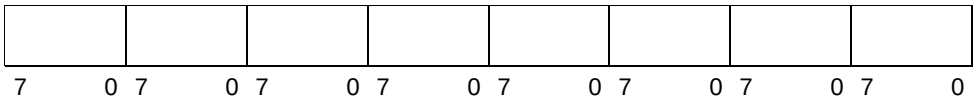
The internal representation of an array type arranges the elements of an array in the form that satisfies the alignment condition (alignment) of the elements

Example

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

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

Figure 4.5 Internal Representation of Array Type



(6) Structure type

(a) Internal representation

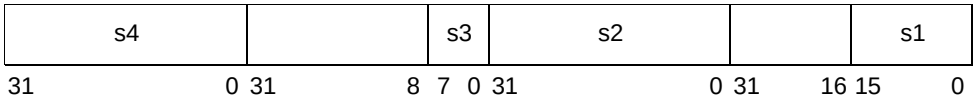
The internal representation of a structure type arranges the elements of a structure in a form that satisfies the alignment condition of the elements.

Example

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

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

Figure 4.6 Internal Representation of Structure Type



For the internal representation when the structure type packing function is used, see "4.2.5.8 Structure type packing".

(7) Union type

(a) Internal representation

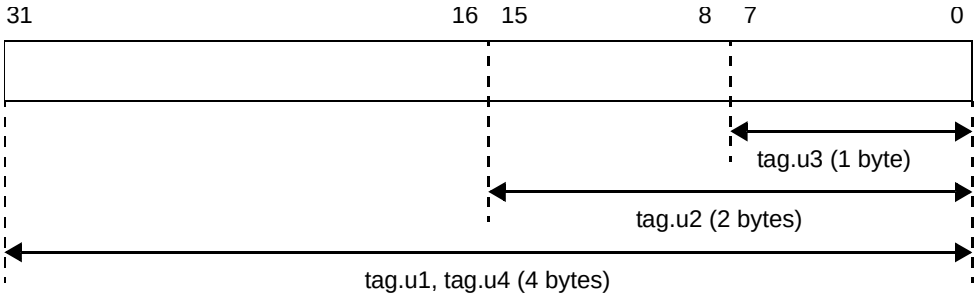
A union is considered as a structure whose members all start with offset 0 and that has sufficient size to accommodate any of its members. The internal representation of a union type is like each element of the union is placed separately at the same address.

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 4.7 Internal Representation of Union Type



(8) Bit field

(a) Internal representation

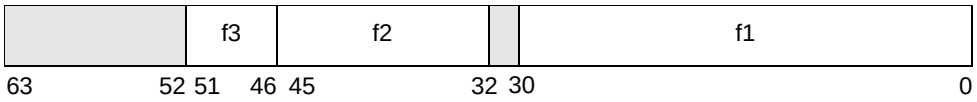
The most significant bit of a bit field declared as a signed type, or without an explicit sign declaration, will be the sign bit. The first bit field to be declared will be allocated starting from the least significant bit in the area with the sign of the type when the bit field was declared. If the 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 "4.2.5.9 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 4.8 Internal Representation of Bit Field

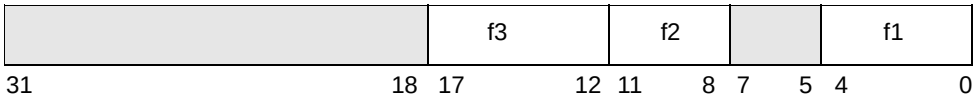


Example 2.

```
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 4.9 Internal Representation of Bit Field



The ANSI standards do allow only int and unsigned int types to be specified for a bit field, but the CC-RH allows char, short, long, long long and those unsigned types.
For the internal representation of bit field when the structure type packing function is used, see "4.2.5.8 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 4.11 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.

4.1.7 Section name

The following table lists the names, section types, and section attributes of these reserved sections.

Table 4.12 Reserved Sections

Section Type	Default Section Name	Description	Alignment Size
Program areas	.text	Executable program	2

Section Type	Default Section Name	Description	Alignment Size
Uninitial-ized data areas	.bss	Sections accessed with r0-relative ld/st (two instructions)	4
	.zbss	Sections that can be accessed with r0-relative ld/st instruction Generation when #pragma section r0_disp* is specified	
	.zbss23		
	.ebss	Sections that can be accessed with ep-relative ld/st instruction Generation when #pragma section ep_disp* is specified	
	.ebss23		
	.tbss4	Sections that can be accessed with ep-relative sld/sst instruction Generation when #pragma section ep_disp* is specified	
	.tbss5		
	.tbss7		
	.tbss8		
	.sbss	Sections that can be accessed with gp-relative ld/st instruction Generation when #pragma section gp_disp* is specified	
	.sbss23		
Initialized data areas	.data	Sections accessed with r0-relative ld/st (two instructions)	4
	.zdata	Sections that can be accessed with r0-relative ld/st instruction Generation when #pragma section r0_disp* is specified	
	.zdata23		
	.edata	Sections that can be accessed with ep-relative ld/st instruction Generation when #pragma section ep_disp* is specified	
	.edata23		
	.tdata4	Sections that can be accessed with ep-relative sld/sst instruction Generation when #pragma section ep_disp* is specified	
	.tdata5		
	.tdata7		
	.tdata8		
	.sdata	Sections that can be accessed with gp-relative ld/st instruction Generation when #pragma section gp_disp* is specified	
	.sdata23		
Constant data areas	.const	Sections accessed with r0-relative ld/st (two instructions)	4
	.zconst	Sections that can be accessed with r0-relative ld/st instruction Generation when #pragma section .zconst* is specified	
	.zconst23		

The default section name can be changed using #pragma section.

4.1.8 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.

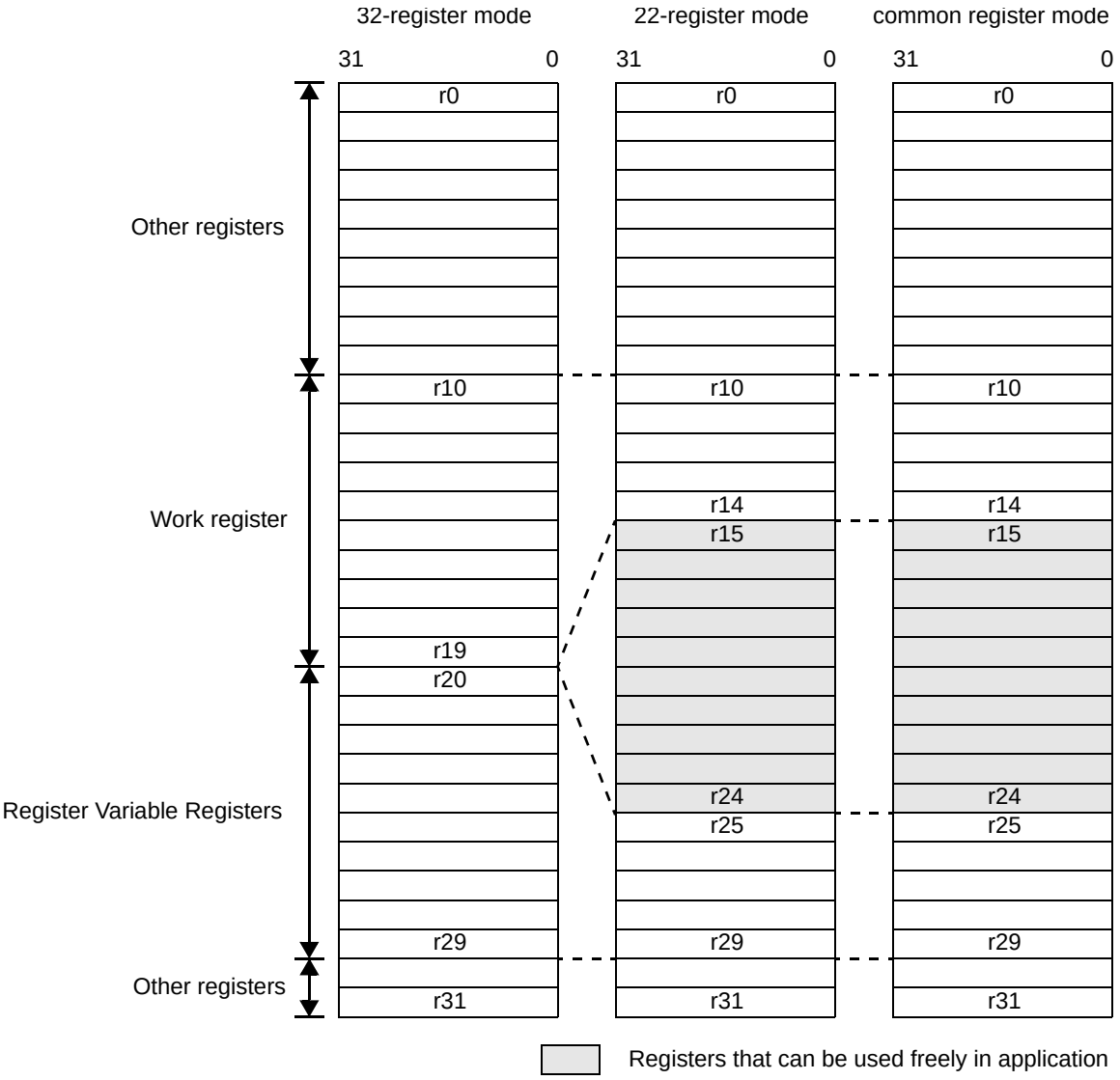
Caution In an application program that has many variables to be allocated to registers by the CC-RH, the variables so far allocated to a register are accessed from memory when a register mode has been specified. As a result, the processing speed may drop.

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

Table 4.13 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 4.10 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 Macro name

Below are #pragma directives supported as extended language specifications.

Table 4.14 List of Supported Macros

Macro Name	Definition
__LINE__	Line number of source line at that point (decimal).
__FILE__	Name of source file (character string constant).
__DATE__	Date of translating source file (character string constant in the form of "Mmm dd yyyy"). Here, the name of the month is the same as that created by the asctime function stipulated by ANSI standards (3 alphabetic characters with only the first character is capital letter) . The first character of dd is blank if its value is less than 10.
__TIME__	Translation time of source file (character string constant having format "hh:mm:ss" similar to the time created by the asctime function).
__STDC__	Decimal constant 1 (defined when the -Xansi option is specified).[Note]
__RENESAS__	This value is not set.
__RENESAS_VERSION__	If the version is V.XX.YY.ZZ, this will be 0xXXYYZZ00. Example) V.1.00.00 -> -D__RENESAS_VERSION__=0x01000000
__CCRH__	This value is not set .
__CCRH	This value is not set .
__CC850__	This value is not set.
__CC850	This value is not set.
__v850e3v5__	This value is not set (defined when v850e3v5 or rh850 is specified by the -Xcommon option is specified).
__v850e3v5	This value is not set (defined when v850e3v5 or rh850 is specified by the -Xcommon option is specified).
__DBL8	This value is not set .
__DOUBLE_IS_64BITS__	This value is not set.
__BITLEFT	This value is not set (defined when left is specified by the -Xbit_order option).
__BITRIGHT	This value is not set (defined when right is specified by the -Xbit_order option).
__AUTO_ENUM	This value is not set (defined when auto is specified by the -Xenum_type option).
__FPU	This value is not set (defined when fpu is specified by the -Xfloat option).
__CHAR_SIGNED__	This value is not set.).
Register mode macro	Since this macro indicates the target CPU, its value is not set. Macro defined with register mode is as follows. 32 register mode: __reg32__ 22 register mode: __reg22__ Universal register mode: __reg_common__

Macro Name	Definition
_LIT	This value is not set.
__MULTI_LEVEL__	level value designated in Xmulti_level = level

Note For the processing to be performed when the -Xansi option is specified, see "4.1.5 Option to process in strict accordance with ANSI standard".

4.2.2 Reserved words

The CC-RH adds the following characters as reserved words to implement the expanded function. These words are similar to the ANSI C keywords, and cannot be used as a label or variable name.

Reserved words that are added by the CC-RH are listed below.

_Bool, __bsh, __bsw, __caxi, __clr1, __di, __ei, __halt, __hsw, inline, __ldlw, __ldsr, __mul32, __mul32u, __nop, __not1, __satadd, __satsub, __sch0l, __sch0r, __sch1l, __sch1r, __set1, __set_il, __stcw, __stsr, __synce, __synci, __syncm, __syncp

All names that include two underscores (__) are also invalid as label or variable names.

4.2.3 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 4.15 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]
Character string	.STR.*num*
switch table	.SWITCH.LABEL.*num1_num2*[Note].SWITCH.LABEL.*num1_num2*[Note].END

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

4.2.4 #pragma directive

Below are #pragma directives supported as extended language specifications.

Table 4.16 List of Supported #pragma Directive

#pragma directive	Definition
#pragma section *attribute-strings* "*section-name*"	Data or program memory allocation
#pragma section *attribute-strings*	
#pragma section [*character string*]	
#pragma inline_asm [()*function-name*[(size=*numerical value*)][,...]()]	Description with assembler instruction

#pragma directive	Definition
#pragma inline [()*function-name*[, *function-name* ...][)]	Inline expansion
#pragma noinline [()*function-name*[, *function-name*, ...][)]	
#pragma interrupt [()*function-name*[(*interrupt specification* [, *interrupt specification*]...)] [,...] [)]	Interrupt/Exception handler specification
#pragma block_interrupt [()*function-name*[)]	Interrupt disable function specification
#pragma pack [()[1|2|4][)]	Structure type packing specification
#pragma bit_order [{left|right}]	Bit field assignment
#pragma pmodule *pm-specification*	Core number specification (for a multi-core device)

(1) Data or program memory allocation
Allocates variables to an arbitrary section.
For details about the allocation method, see "4.2.5.1 Allocation of data and program to section".

(2) Description with assembler instruction
Assembler directives can be described in a C source program.
For the details of description, see "4.2.5.2 Describing assembler instruction".

(3) Inline expansion

(a) Inline expansion specification
A function that is expanded inline can be specified.
For the details of expansion specification, see "4.2.5.3 Inline expansion".

(b) Specifying functions ineligible for inline expansion
You can specify that a function is not eligible for inline expansion.
See "4.2.5.3 Inline expansion" for details about specifying functions to be ineligible for inline expansion.

(4) Interrupt/Exception handler specification
Interrupt/Exception handlers are described in C language.
For details, see "(3) Describing interrupt/exception handler".

```
#pragma interrupt   [()function-name[(interrupt specification [,interrupt specifi-
cation]...)] [,...] [)]
```

(5) Interrupt disable function specification
Interrupts are disabled for the entire function.
For details, see "(2) Disabling interrupts in entire function".

```
#pragma block_interrupt [()function-name[)]
```

(6) Structure type packing specification
Specifies the packing of a structure type. The packing value, which is an alignment value of the member, is specified as the numeric value. A value of 1, 2, 4 can be specified. When the numeric value is not specified, the setting is the default alignment.
For details, see "4.2.5.8 Structure type packing".

```
#pragma pack [()[1|2|4][)]
```

(7) Bit field assignment
CC-RH can switch the order of a bit field.Specifies the switching the order of a bit field.
For details, see "4.2.5.9 Bit field assignment".

```
#pragma bit_order [{left|right}]
```

(8) Core number specification (for a multi-core device)
A function to which a core number is to be assigned can be specified.
For details, see "4.2.5.10 Core number specification (for a multi-core device)".

```
#pragma pmodule pm-specification
```

4.2.5 Using extended language specifications

This section explains using expanded specifications.

- Allocation of data and program to section
- Describing assembler instruction
- Inline expansion
- Controlling interrupt level
- Interrupt/Exception processing handler
- Disabling or enabling maskable interrupts
- Embedded functions
- Structure type packing
- Bit field assignment
- Core number specification (for a multi-core device)

4.2.5.1 Allocation of data and program to section

CC-RH can allocate modules and data to arbitrary sections at the C-language level. This makes it possible to specify placement using the relocation attribute characteristics of each section.

(1) #pragma section directive
Describe the #pragma section directive in the following format.

```
#pragma section attribute-strings "section-name"
Variable declaration/definition
```

```
#pragma section attribute-strings
Variable declaration/definition
```

```
#pragma section [character string]
Variable declaration/definition
```

Section relocation attributes are specified using attribute strings, and section names are specified in the form "section-name". All static variables included after this pragma will be placed in the specified section in accordance with the section relocation attribute.

If "attribute string" in the format "#pragma section *attribute-string*" is identical to "character string" in the format "#pragma section [*character-string*]", then it will be treated as the format "#pragma section *attribute-string*" format, without outputting a message. However, if the -check=shc option is specified, then a message will be output if treated as the "#pragma section *attribute-string*" format.
The identity test of attribute-string and character-string is case sensitive. To give an example, the all-caps string R0_DISP32 will not be treated as an attribute string.

Example

```
#pragma section r0_disp32   /*Here, "data" matches an attribute string, so it is*/
                            /*seen as the format "#pragma section*/
                            /*attribute-string"*/
#pragma section R0_DISP32   /*This does not match the case of an attribute*/
                            /*string, and so it is treated as the format*/
                            /*"#pragma section [character-string]"*/
```

If the section name starts with a number (0-9), an underscore ("_") is prepended to the section name.

Example #pragma section 123 -> Section name : _123.bss, _123.const, _123.text, ...

The following characters can be used in the character string.

- 0 - 9
- a - z, A - Z
- _
- @
- .

If #pragma section is specified for either a declaration or a definition, the one specified with #pragma section is valid.
If different #pragma section directives are specified for declarations or definitions, the #pragma section that appears first is valid.

(a) Specifying "#pragma section *attribute-string* "*section-name*""
Renames the default section, and places variables declared after the #pragma in the new section name.

- This pragma directive renames the default section to the name specified by the user.
- This pragma directive is valid for external variables, string literals, function-internal static variables, and functions.
- This pragma directive invalidates specifications of the same attribute coded prior to this point, and enables the new section specification. If no new section is specified, then it is valid to the end of the file.
- If default is specified as the relocation attribute, then subsequent assignments revert to the default with no #pragma section declared. "default" disables all #pragma section statements.
- If there is no variable initialization, then it is placed in the bss section corresponding to the specified relocation attribute
- By default, string literals are assigned to .const. The placement for #pragma section declarations is the same as for other const objects.
- A section name consists of the user-specified string, followed automatically by a string representing the relocation attribute. Thus depending on whether initialization is enabled, if the relocation attribute is switched automatically, it is possible that a section with the same name will be produced.
  Example: #pragma section r0_disp32 "xxx" -> Section name : xxx.data
- If a const relocation attribute is specified, then only variables with the const modifier are allocated to the section, and variables without this modifier are allocated to the default section.

Table 4.17 Section Relocation Attribute

Target	Renamable Attribute Strings	Default Section Name	User-modified Section Name-Note
Definition (type data)	r0_disp16	.zdata	*xxx*.zdata
	r0_disp23	.zdata23	*xxx*.zdata23
	r0_disp32	.data	*xxx*.data
	ep_disp4	.tdata4	*xxx*.tdata4
	ep_disp5	.tdata5	*xxx*.tdata5
	ep_disp7	.tdata7	*xxx*.tdata7
	ep_disp8	.tdata8	*xxx*.tdata8
	ep_disp16	.edata	*xxx*.edata
	ep_disp23	.edata23	*xxx*.edata23
	gp_disp16	.sdata	*xxx*.sdata
	gp_disp23	.sdata23	*xxx*.sdata23

Target	Renamable Attribute Strings	Default Section Name	User-modified Section Name-Note
Declaration (type bss)	The compiler automatically determines variable definitions without an initial value to be of type bss. It is thus not a bss attribute string.	.zbss .zbss23 .bss .tbss4 .tbss5 .tbss7 .tbss8 .ebss .ebss23 .sbss .sbss23	*xxx*.zbss *xxx*.zbss23 *xxx*.bss *xxx*.tbss4 *xxx*.tbss5 *xxx*.tbss7 *xxx*.tbss8 *xxx*.ebss *xxx*.ebss23 *xxx*.sbss *xxx*.sbss23
Type const	const zconst zconst23	.const .zconst .zconst23	*xxx*.const *xxx*.zconst *xxx*.zconst23
Type text	text	.text	*xxx*.text
Clearing of specification	default	Clears all specifications, and restores the default section name.	

Note This is the actual section name when the section name of "xxx" is given.

Example 1.

```
#pragma section r0_disp32   "mydata"
int i = 0;                      /*Allocated to "mydata.data"*/
int j;                          /*Allocated to "mydata.bss"*/

#pragma section gp_disp16   "mydata2"
int i2 = 1;                     /*Allocated to "mydata2.sdata"*/

#pragma section const       "myconst"
const int c2 = 0;               /*Allocated to "myconst.const"*/
int j2;                         /*Allocated to ".bss"*/
```

Example 2.

```
#pragma section ep_disp4    "XXX"
extern  int     i;
void func()
{
        i = 5;
}
#pragma section r0_disp32   "OOO"
int i = 5;              /*Allocated to "XXX.tdata4"*/
                        /*When #pragma section is specified for the first*/
                        /*declaration, it takes priority*/
```

Example 3.

```
#pragma section ep_disp4
int     i;
#pragma section ep_disp23
int     i;              /*Allocated to "tbss4"*/
                        /*When #pragma section is specified for the first*/
                        /*declaration, it takes priority*/
```

Example 4.

```
<def.c>
#pragma section ep_disp4
extern  int     i;
#pragma section ep_disp4    "XXX"
int i = 5;                      /*Allocated to ".tdata4"*/
                                /*When #pragma section is specified for the*/
                                /*first declaration, it takes priority*/

<use.c>
#pragma section ep_disp4
extern  int     i;
void func()
{
        i = 5;                  /*No problems with access*/
}
```

Example 5.

```
#pragma section ep_disp4    "XXX"
extern  int     i;
#pragma section ep_disp4     "OOO"
int     i = 5;                  /*Allocated to ".tdata4"*/
                                /*When #pragma section is specified for the*/
                                /*first declaration, it takes priority*/
```

Example 6.

```
#pragma section r0_disp16   "myzdata"
#pragma section const       "myconst"   /*Specification of "myzdata" ends*/
                                        /*at this line*/
int     i = 0;                          /*No effect, because not .data const*/
const int       ci = 0;                 /*myconst.const*/
#pragma section r0_disp16   "myzadata"  /*Re-specify*/
int     j = 0;                          /*Becomes myzdata.zdata*/
const int       cj = 0;                 /*The myconst specification in the*/
                                        /*pragma immediately preceding the*/
                                        /*.const is erased, and it is*/
                                        /*placed in ".const".*/
```

Example 7.

```
extern  int     i;

#pragma section ep_disp4
int     i;      /*Allocated to ".tdata4"*/
                /*Although this differs from the declaration side, even if*/
                /*this is accessed without #pragma section ep_disp23 in*/
                /*other translation units, access is correctly done in*/
                /*long-distance addressing mode (however, it is not*/
                /*efficient)*/
```

Example 8.

```
#pragma section ep_disp23
#pragma section default
extern  int     i;

#pragma section ep_disp4
int     i;      /*Allocated to ".tdata4"*/
                /*#pragma section default, which restores the section name*/
                /*to the default name, is not regarded as a section*/
                /*specification*/
                /*The extern declaration is handled as a declaration*/
                /*without section specification*/
```

(b) Specifying "#pragma section *attribute-strings*"
Variables declared after the #pragma will be placed in the default section.

- Specifies placement of data in the default section name.
- The placement rules are the same as for "#pragma section *attribute-string* "*section-name*"", except for the fact that they are placed in the default section name.

Table 4.18 Section Relocation Attribute

Target	Attribute Strings	Default Section Name
Definition (type data)	r0_disp16	.zdata
	r0_disp23	.zdata23
	r0_disp32	.data
	ep_auto	Automatically selected from among .tdata4, .tdata5, .tdata7, and .tdata8
	ep_disp4	.tdata4
	ep_disp5	.tdata5
	ep_disp7	.tdata7
	ep_disp8	.tdata8
	ep_disp16	.edata
	ep_disp23	.edata23
	gp_disp16	.sdata
	gp_disp23	.sdata23
Declaration (type bss)	The compiler automatically determines variable definitions without an initial value to be of type bss. It is thus not a bss attribute string.	.zbss
		.zbss23
		.bss
		.tbss4
		.tbss5
		.tbss7
		.tbss8
		.ebss
		.ebss23
		.sbss
		.sbss23
Type const	const	.const
	zconst	.zconst
	zconst23	.zconst23
Type text	text	.text

Target	Attribute Strings	Default Section Name
Clearing of specification	default	Clears all specifications, and restores the default section name.

Example

```
#pragma section gp_disp16
int a = 1;                      /*Allocated to ".sdata"*/
int b;                          /*Allocated to ".sbss"*/
```

- -ep_auto produces the same behavior as ep_disp4, ep_disp5, ep_disp7, or ep_disp8 depending on the type of declaration.

Type of Declaration	Corresponding ep_disp*n*
unsigned char	ep_disp4
unsigned short	ep_disp5
signed char char _Bool	ep_disp7
signed short signed int unsigned int signed long unsigned long signed long long unsigned long long float double long double pointer enumerate type (when the size of the elements is that of the int type)	ep_disp8

Remark 1. The ep_disp*n* for array-type variables depends on the type of the elements. In a multi-dimensional array (an array of arrays), however, the selected ep_disp*n* corresponds to the type of the elements stored in the array.

```
#pragma section ep_auto
char    c_ary[3][5] = {0};      /*.tdada7 (char)*/
char*   p_ary[3][5] = {0};      /*.tdata8 (pointer)*/
```

Remark 2. If a variable of the enumerate type is smaller than one of the int type due to the -Xenum_type option having been specified, the ep_disp*n* that corresponds to the given smaller type is selected.

Remark 3. Structures and unions are not allocated as any from among .tdata4, .tdata5, .tdata7, and .tdata8.

- Since ep_auto produces the same behavior as ep_disp4, ep_disp5, ep_disp7, or ep_disp8 depending on the type of declaration, ep_auto and the ep_disp*n* that produces the same behavior being specified for the declaration and definition of the same variable does not lead to an error. ep_auto and an ep_disp*n* that produces different behavior being specified for the declaration and definition of the same variable will lead to an error.

Remark 1.

```
#pragma section ep_auto
extern  unsigned char   ch;
#pragma section ep_disp4
unsigned char       ch; /*Allocated as .tdata4 with no errors or warnings*/
```

Example 1.

```
#pragma section ep_auto
extern  unsigned char   ch;
#pragma section ep_disp7
unsigned char       ch; /*Allocated as .tdata4 with no errors or warnings*/
                        /*When #pragma section is specified for the first*/

                        /*declaration, it takes priority*/
```

(c) Specifying "#pragma section [*character-string*]"
This switches the section name output by the compiler.
The section name after switching from the default section name is as follows.

Target	Specifying	Default	After Switching
Program	#pragma section *xxx*	text	*xxx*.text
Constant		const	*xxx*.const
Initialized data		data	*xxx*.data
Uninitialized data		bss	*xxx*.bss

- If the string is omitted, all section names are set to the default.

Example

```
#pragma section abc
int a;                  /*a is allocated to the section abc.bss*/
const int d=1;          /*d is allocated to the section abc.const*/
void f(void)            /*f is allocated to the section abc.text*/
{
        a = d;
}

#pragma section
int b;                  /*b is allocated to the section .bss*/
void g(void)            /*g is allocated to the section .text*/
{
        b = d;
}

#pragma section 123
int c;                  /*c is allocated to the section _123.bss*/
void h(void)            /*h is allocated to the section _123.text*/
{
        c = d;
}
```

4.2.5.2 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.

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(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.
- If you write a label in a function coded in assembly, labels with the same name will be created for each inline expansion.
  In this situation, use local labels coded in assembly. Although local labels have the same name in the assembly source, the assembler converts them to different names automatically.

4.2.5.3 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, ...][)]
```

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) The addresses of formal parameters are not referenced from inside the function.
If you reference the address of a formal parameter from inside a function, then the inline expansion specification will be ignored, without outputting an error or warning message.

(h) 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.

(i) An interrupt handler is not specified to be expanded inline.
A function specified by the #pragma interrupt is recognized as an interrupt handler. If inline expansion is specified for this function, the CC-RH outputs a warning message and ignores the specification.

(j) Interrupts are not disabled in a function by the #pragma block_interrupt directive.
#If inline expansion is specified for a function in which interrupts are declared by the #pragma block_interrupt directive to be disabled, the CC-RH outputs a warning message and ignores the specification.

(k) If #pragma inline_asm is specified, then a #pragma inline specification will be ignored.
If you specify inline expansion for a function coded in assembly using #pragma inline_asm, then a warning message will be output, and the #pragma inline specification will be ignored.

(l) If you specify -Xmerge_files, then functions may be inlined even if they are not coded within the file.

(3) Functions ineligible for inline expansion
When using the -Oinline option, use #pragma noinline to prevent inline expansion of a specific function.

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

Specifying #pragma inline and #pragma noinline for the same function simultaneously within a single translation unit will cause an error.

(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;
}
```

4.2.5.4 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 register);
```

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

4.2.5.5 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-name[(interrupt specification [, interrupt speci-
fication]...)] [,...] [)]
```

Describe functions that are described in the C language. In the case of a function, "void func1() {}", specify "func1".
Always give interrupt functions a return type of void.
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.

- Define an interrupt function
  You can define an interrupt function with either no parameters or one parameter.

  Example 1. Using no parameters

```
void i_func(void)
{
        :
}
```

  Example 2. Using one parameter

```
void i_func(unsigned long dummy)
{
        :
}
```

- 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/HVTRAP/FETRAP/TRAP0/TRAP1/RIE/FPP/FPI/UCPOP/MIP/MDP/ITLBE/DTLBE/PIE/MAE/FENMI/FEINT/EINT_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. 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 option is specified. Interpreted as false when -Xfloat=soft 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.

If you write an interrupt specification, you must include a term to the right of the equals sign ("=").
For example, writing only "enable=" will cause a compilation error. The default interrupt specification signifies the behavior when individual interrupt specifications are not written.

- 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.

- [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

- [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    r9, 6               ; (9)
    ldsr    r8, 7               ; (9)
    ld23.dw 0x00000008[r3], r8  ; (10)
    ldsr    r9, 17              ; (10)
    ldsr    r8, 16              ; (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    r7, 1               ; (12)
    ldsr    r6, 0               ; (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    r9, 6               ; (9)
    ldsr    r8, 7               ; (9)
    ld23.dw 0x00000008[r3], r8  ; (10)
    ldsr    r9, 17              ; (10)
    ldsr    r8, 16              ; (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    r7, 1               ; (12)
    ldsr    r6, 0               ; (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

- 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.

  - [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

  - [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    r9, 6               ; (6)
    ldsr    r8, 7               ; (6)
    ld23.dw 0x00000008[r3], r8  ; (7)
    ldsr    r9, 17              ; (7)
    ldsr    r8, 16              ; (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

- 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.

  - [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.
  - [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.

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

(4) Notes on describing interrupt/exception handler

- A function specified as an interrupt/exception handler cannot be expanded inline. The #pragma inline directive is ignored even if specified.

4.2.5.6 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[)]
```

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 an interrupt handler and a #pragma block_interrupt directive are specified for the same interrupt, the interrupt handler takes precedence, and the setting of disabling interrupts is ignored.
- 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.
- Neither #pragma inline nor inline expansion can be specified by an optimization option for the function specified by a #pragma block_interrupt directive. The inline expansion specification is ignored.
- 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.

4.2.5.7 Embedded functions

In the CC-RH, some of the assembler instructions can be described in C source as "Embedded 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 embedded function, then an warning is output, and it is treated as an embedded function.

A error is also output if a register number that does not exist in the hardware is specified for ldsr()/stsr().

The instructions that can be described as functions are as follows.

Table 4.20 Assembler Instruction

Assembler Instruction	Function	Embedded Function
di	Interrupt control	void __DI(void);
ei		void __EI(void);

-	Interrupt-priority-level control	int NUM;
		void* ADDR[Note 1];
		void __set_il_rh(NUM, ADDR);
		
		- 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 a, b; long __satadd(a, b);
satsub	Saturated subtraction	long a, b; long __satsub(a, b);
bsh	Halfword data byte swap	long a; long __bsh(a);
bsw	Word data byte swap	long a; long __bsw(a);
hsw	Word data halfword swap	long a; long __hsw(a);
mul	Instruction that assigns higher 32 bits of multiplication result to variable	long a, b; long __mul32(a, b);

Assembler Instruction	Function	Embedded Function
mulu	Instruction that assigns higher 32 bits of unsigned multiplication result to variable	unsigned long a, b; unsigned long __mul32u(a, b);
sch0l	Bit (0) search from MSB side	long a; long __sch0l(a);
sch0r	Bit (0) search from LSB side	long a; long __sch0r(a);
sch1l	Bit (1) search from MSB side	long a; long __sch1l(a);
sch1r	Bit (1) search from LSB side	long a; long __sch1r(a);
ldsr	Loads to system register	long regID; unsigned long a; void __ldsr(regID[Note 2], a);
ldsr	Loads to system register	long regID; long selID; unsigned long a; void __ldsr_rh(regID[Note 2], selID[Note 2], a);
stsr	Stores contents of system register	long regID; unsigned long __stsr(regID[Note 2]);
stsr	Stores contents of system register	long regID; long selID; unsigned long __stsr_rh(regID[Note 2], selID[Note 2]);
caxi	Compare and Exchange	long *a; long b, c; long __caxi(a, b, c);
clr1	Bit clear	unsigned char *a; int bit; void __clr1(a, bit);
set1	Bit set	unsigned char *a; int bit; void __set1(a, bit);
not1	Bit not	unsigned char *a; int bit; void __not1(a, bit);
ldl.w[Note 3]	Atomic load.	long *a; long __ldlw(a);
stc.w[Note 3]	Store	long *a; long b; void __stcw(a, b);
synce	Exception synchronization	void __synce(void);
synce	Instruction pipeline synchronization	void __synci(void);
syncm	Memory synchronization	void __syncm(void);

Assembler Instruction	Function	Embedded Function
syncp	Pipeline synchronization	void __syncp(void);

Note 1. For ADDR, specify the address of the interrupt control register.

Note 2. Specified the system register number (0 to 31) in regID and 0 to 31 in selID.

Note 3. A warning is output when the -Xcpu=g3k option is used.

Caution Even if a function is defined with the same name as an embedded function, it cannot be used.
If an att isempt made to call such a function, processing for the embedded function provided by the compiler takes precedence.

4.2.5.8 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][)]
```

#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

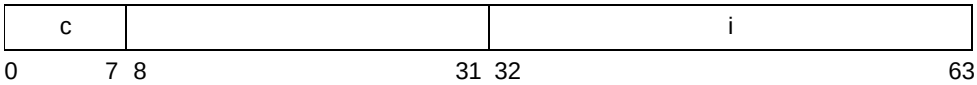
```
#pragma pack 1  /*structure member aligned using 1-byte alignment*/
struct  TAG {
        char    c;
        int     i;
        short   s;
};
```

(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.

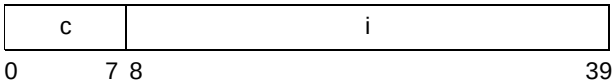
Example

```
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*/
```

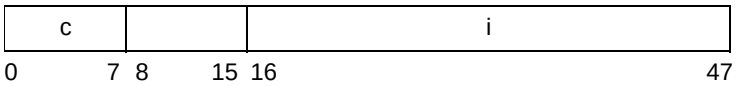
sobj



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.

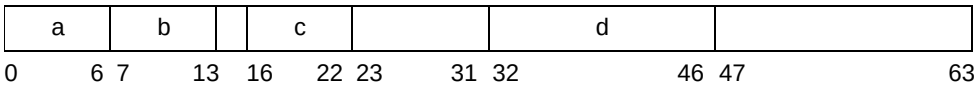
(b) When the structure type packing value is smaller than the alignment condition value of the element type

- 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.

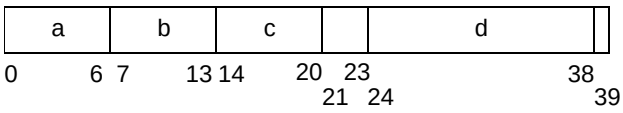
Example

```
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



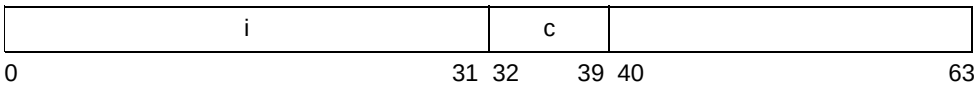
(5) Alignment condition of top structure object
The alignment condition of the top structure object is the same as when the structure packing function is not used.

(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. The alignment condition of the top structure object is the same as when the structure packing function is not used.

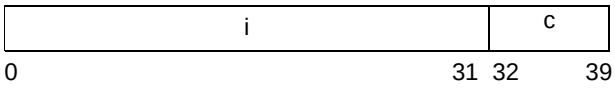
Example 1.

```
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*/
```

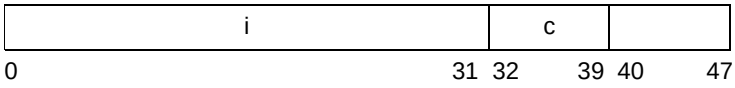
sobj



s1obj



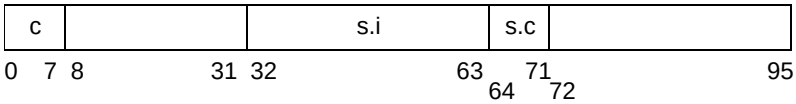
s2obj



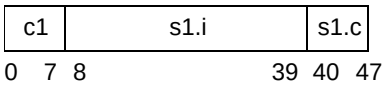
Example 2.

```
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*/
```

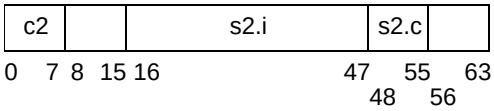
tobj



t1obj



t2obj



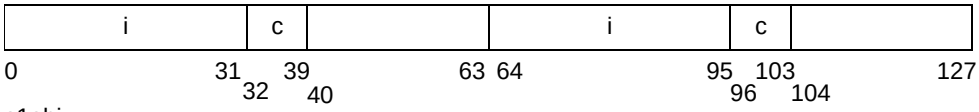
(7) Size of structure array

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.

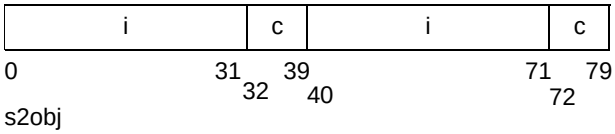
Example

```
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*/
```

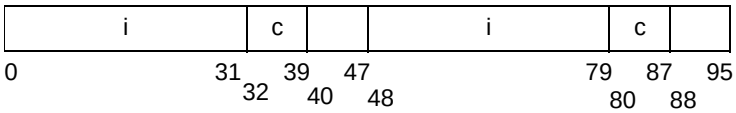
sobj



s1obj



s2obj



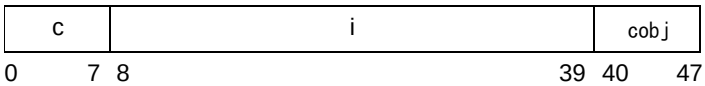
(8) Area between objects

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 pack=1 or #pragma pack 1 is specified for a structure, a union, or a class, its members cannot be accessed by using a pointer (including access using a pointer within a member function).

Example

```
#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. */
}
```

4.2.5.9 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}]
```

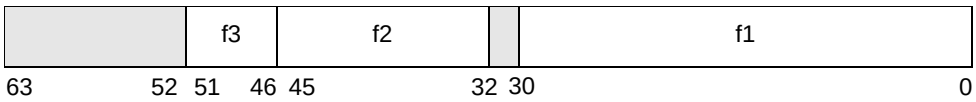
If left is specified, then members are assigned from the MSB; if right is specified, then they are assigned from the LSB.

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 4.11 Internal Representation of Bit Field

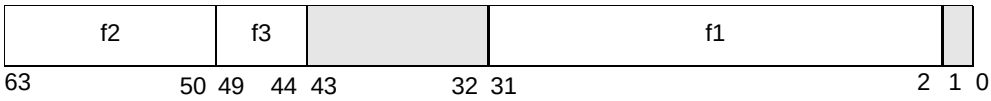


Example 2.

```
#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 4.12 Internal Representation of Bit Field

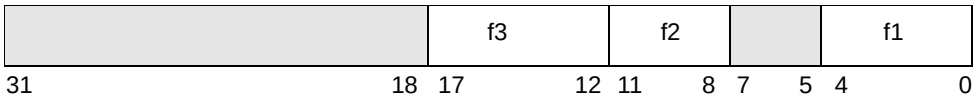


Example 3.

```
#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 4.13 Internal Representation of Bit Field

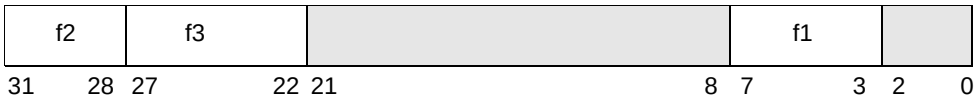


Example 4.

```
#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 4.14 Internal Representation of Bit Field



4.2.5.10 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.

- 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.

- 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.
- The compiler can check the correspondence between core numbers and data or functions (If a function that should be executed only in core 1 calls a function that should be executed only in core 2, the compiler can detect it as an error).

(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 option is specified. If the -Xmulti_level 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
	pm1	Data or function for core 1	***.pm1
	:	:	:
	pm255	Data or function for core 255	***.pm255
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 cmn or pm1 to pm255 is written, a compilation error is generated.
- #pragma pmodule is applied to all static variable declarations, function declarations, and string literals[Note] that appear after the #pragma pmodule declaration line.

  Note This directive is applied to all items allocated to the section; it is also applied to the string literals allocated to the const section.

- #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 the same variable or function declaration is specified multiple times within a single translation unit, if different #pragma pmodule specifications are written for them, the first specification is valid.

  Example

```
#pragma pmodule pm1
extern  int     i;
#pragma pmodule pm2
int     i = 5;          //pm1 is valid
```

The following shows specification examples.
These examples assume that the -Xmulti_level=1 is specified.

Example 1.

```
#pragma section r0_disp16
int     i;                      //.zbss.pm1
--------------------------------------------------------------------------------
#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
--------------------------------------------------------------------------------
#pragma pmodule pm2
#pragma pmodule pm3
#pragma section r0_disp16
int     i;                      //.zbss.pm3
#pragma pmodule pm1
int     j;                      //.zbss.pm1
```

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 pm2
extern  int     i;
#pragma pmodule pm3
int     i;                      //.bss.pm2 (No warning)
```

4.2.6 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 ANSI. 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

```
#ifndef __CCRH__
#define interrupt       /*considered interrupt function as normal function*/
#endif
```

Example 2. Change to other type

```
#ifdef  __v850e3v5__
#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, expressio, and operators.

5.1.1 Description

An assembly language statement consists of a "symbol", a "mnemonic", "operands", and a "comment".

```
[symbol][:Δ]        [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.

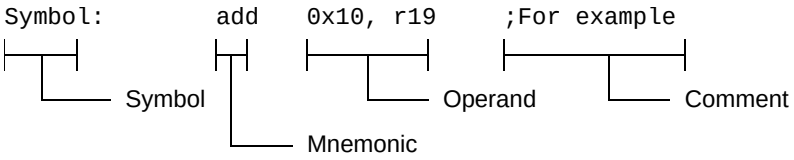
It is irrelevant whether blanks are inserted in the following location.

- 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 an identifier and constant
@	Constitutes an identifier and constant

Character	Usage
_ (underscore)	Constitutes an identifier
.(period)	Constitutes an identifier and constant
~	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 and symbol indicating a complete macro parameter
<	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
=	Relational operator
!	Beginning immediate addressing and negation operator
Δ (blank)	Field delimiter
~	Concatenation symbol (in macro body)
&	Logical product operator
#	Beginning indicates and comment
[ ]	Indirect indication symbol
"(double quotation)	Start and end of character string constant
%	ep offset referring 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 except 0x00 (including multibyte kanji, although the encoding depends on the OS). If 0x00 is encountered, an error occurs and all characters from the 0x00 to the closing single quote (') are ignored.

(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). If you wish to specify a number as first character with a section name, enclose each file name with a double quotation (").
- 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.
```

(c) Points to note about symbols

The assembler generates a name automatically when a section definition directive does not specify a name.
These section names are listed below.
Duplicate section name definitions are errors.

Section Name	Directive	Relocation Attribute
.text	.cseg directive	TEXT
.const		CONST
.zconst		ZCONST
.zconst23		ZCONST23
.bss	.dseg directive	BSS
.data		DATA
.sbss		SBSS
.sdata		SDATA
.sbss23		SBSS23
.sdata23		SDATA23
.tdata		TDATA
.tbss4		TBSS4
.tdata4		TDATA4
.tbss5		TBSS5
.tdata5		TDATA5
.tbss7		TBSS7
.tdata7		TDATA7
.tbss8		TBSS8
.tdata8		TDATA8
.ebss		EBSS
.edata		EDATA
.ebss23		EBSS23
.edata23		EDATA23
.zbss		ZBSS
.zdata		ZDATA
.zbss23		ZBSS23
.zdata23		ZDATA23

(d) Symbol attributes
Every symbol and label has both a value and an attribute.
The value is the value of the defined data object, for example a numerical value, or the value of the address itself.
Macro names do not have values.
The following table lists symbol attributes.

Attribute Type	Classification	Value
BIT	- Symbols defined as bit values - Symbols defined with the EXTBIT directive	Decimal notation: -2147483648 to 2147483647 Hexadecimal notation: 0x80000000 to 0x7FFFFFFF (signed)
MACRO	Macro names defined with the Macro directive	These attribute types have no values.
FNUMBER	Symbols defined with the FLOAT directive (Single precision floating point)	1.40129846e-45 to 3.40282347e+38
DFNUMBER	Symbols defined with theDOUBLE directive (Double-precision floating point)	4.9406564584124654e-324 to 1.7976931348623157e+308

(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.
However, if the first operand begins with "#", "$","!", or "[", the statement will be assembled properly even if nothing exists between the mnemonic field and the first operand field.

Example Correct mnemonics

```
mov     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" suffix to the number.	0b1101
Octal	Append an "0" suffix to the number.	074
Decimal	Simply write the number.	128

Type	Notation	Example
Hexadecimal	Append an "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

Escape Sequence	Value	Meaning
\?	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 (").
To include the single quote character in the string, write it twice in succession.

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 or location counter (indicated by $), 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	>>, <<
Byte separation operators	HIGH, LOW
2-byte separation operators	HIGHW, LOWW, HIGHW1
Section aggregation 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, !
	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.
  If intermediate values overflow 32 bits, the overflow value is ignored.
- If the value of a constant exceeds 32 bits, an error occurs, and its value is calculated as 0.
- In division, the decimal fraction part is discarded.
  If the divisor is 0, an error occurs and the result is 0.
- Negative values are represented as two's complement.
- External reference symbols are evaluated as 0 at the time when the source is assembled (the evaluation value is determined at link time).

(1) Evaluation examples

Expression	Evaluation
2 + 4 * 5	22
(2 + 3) * 4	20
10/4	2
0 - 1	0xFFFFFFFF
-1 > 1	0x0 (False)
EXT[Note] + 1	1

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:
0xFFFC is returned.
Therefore, (1) can also be described as: mov 0xFFFC, r10.

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 ~0 (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 ~0 (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 ~0(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 ~0 (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 ~0 (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 ~0 (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.

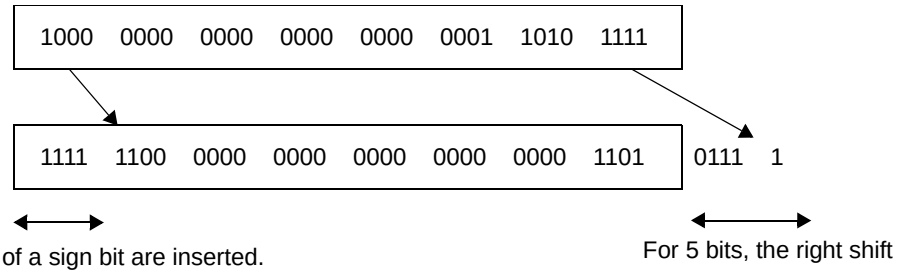
The sign bit is inserted in the high-order bits, the same number of times as the number of bits that were shifted.

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.

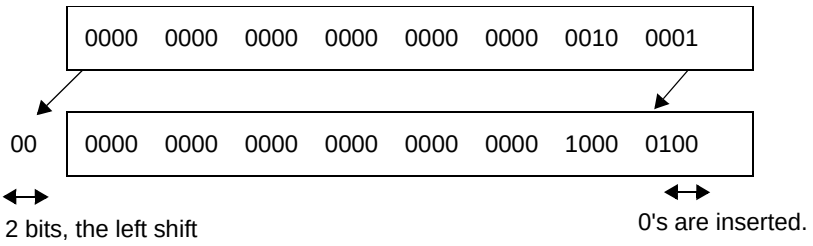
Zeros equivalent to the specified number of bits shifted move into the low-order bits.

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]

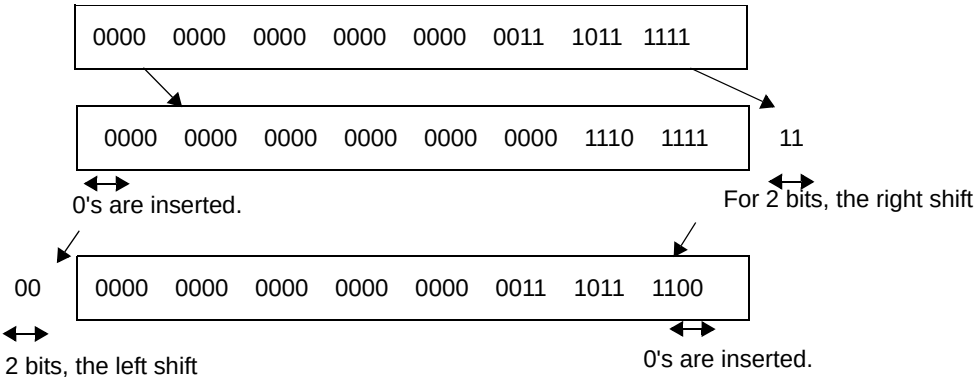
```
mov32   0x21 << 2, r20          ; (1)
```

(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(0x1234), r10       ; (1)
```

(1) By executing a mov instruction, this operator returns the high-order 8-bit value "0x12" of the expression "0x1234". Therefore, (1) in the above example can also be described as: mov 0x12, 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(0x1234), r10        ; (1)
```

(1) By executing a mov instruction, this operator returns the low-order 8-bit value "0x34" of the expression "0x1234". Therefore, (1) in the above example can also be described as: mov 0x34, 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.

[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.

5.1.9 Section aggregation operators

The following section aggregation 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.

Each section and its alignment condition must be specified.

```
.DW         STARTOF(user.text)
.SECTION    user.text, TEXT
.ALIGN      4
```

The alignment condition can be omitted if the default condition for that section is to be used.

```
.DW         STARTOF(user.text)
.SECTION    user.text, TEXT
```

The section definition and alignment condition can be omitted if the default section name and default alignment condition are to be used for that section.

```
.DW         STARTOF(.text)
```

To use this operator in conjunction with SIZEOF:

```
.DW         STARTOF(.data) + SIZEOF(.data)
```

To specify a section name that begins with a number:

```
.DW         STARTOF(123.user.text)
.SECTION    "123.user.text", TEXT
```

[Caution]

- STARTOF can only be written as an operand of the data definition directive, .dw.
- Section names that are not defined in the same module are not specifiable.
  For this reason, if a section is defined in another module, the section definition directive you use to define the section and the .align directive you use to adjust its alignment condition must match those of the definitions of the same section in other modules.
  If a section is specified but has not been defined or has a different alignment condition to that selected in another module, the following messages are output.
  - When the section has not been defined:

    E0550249: Illegal syntax.

  - When the section has a different alignment condition:

    W0561322: Section alignment mismatch : "*section*"

- You can omit the section definitions and .align directives when default section names and default values of the alignment conditions are to be used, respectively.
- 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.

Each section and its alignment condition must be specified.

```
.DW         SIZEOF(user.text)
.SECTION    user.text, TEXT
.ALIGN      4
```

The alignment condition can be omitted if the default condition for that section is to be used.

```
.DW         SIZEOF(user.text)
.SECTION    user.text, TEXT
```

The section definition and alignment condition can be omitted if the default section name and default alignment condition are to be used for that section.

```
.DW         SIZEOF(.text)
```

To use this operator in conjunction with STARTOF:

```
.DW         STARTOF(.data) + SIZEOF(.data)
```

To specify a section name that begins with a number:

```
.DW         SIZEOF(123.user.text)
.SECTION    "123.user.text", TEXT
```

[Caution]

- SIZEOF can only be written as an operand of the data definition directive, .dw.
- Section names that are not defined in the same module are not specifiable.
  For this reason, if a section is defined in another module, the section definition directive you use to define the section and the .align directive you use to adjust its alignment condition must match those of the definitions of the same section in other modules.
  If a section is specified but has not been defined or has a different alignment condition to that selected in another module, the following messages are output.
  - When the section has not been defined:

    E0550249: Illegal syntax.

  - When the section has a different alignment condition:

    W0561322: Section alignment mismatch : "*section*"

- You can omit the section definitions and .align directives when default section names and default values of the alignment conditions are to be used, respectively.
- 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
```

```
(4 + 3) * 2
└─────┘  │
  (1)    │
   └─────┘
     (2)
```

Calculations are performed in the order of expressions (1), (2) and the value "14" is returned as a result.
If parentheses are not used,

```
4 + 3 * 2
│   └──┘
│    (1)
└────┘
  (2)
```

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 an undefined symbol reference 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.

However, if a reference to the absolute address of a label not having a definition in the specified file is specified as label reference on one side of "- label reference" in an "expression related to label reference", it is assumed that the same reference method as that of the label on the other side is used, because of the current organization of the assembler. Note that an absolute expression in this format cannot be specified for a branch instruction. If such an expression is specified, 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).
- 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 5.5 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 5.6 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 5.7 Relocation Attributes of .cseg

Relocation Attribute	Description Format	Explanation	Default Value of Alignment Condition
TEXT	TEXT	Allocates the program.	2
ZCONST	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 instructions using r0 and 16-bit displacement.	4
ZCONST23	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 instructions using r0 and 23-bit displacement.	4
CONST	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.	4

- If there is a label or an instruction to output object code before a section definition directive, a relocatable code section is generated as the default section. The section name here will be ".text", and the relocation attribute will be "TEXT".
- The assembler will output an error if a relocation attribute other than "Table 5.7 Relocation Attributes of .cseg" is 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.

Relocation Attribute	Default Section Name
TEXT	.text
ZCONST	.zconst
ZCONST23	.zconst23
CONST	.const

The default section names have the relocation attributes shown above. Giving them any other attributes is not possible.

- If you wish to specify a number as first character with a section name, enclose each file name with a double quotation (").

  Example

```
test        .cseg   text
            nop
            nop
"test"      .cseg   text
            nop
            nop
"123test"   .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 5.8 Relocation Attributes of .dseg

Relocation Attribute	Description Format	Explanation	Default Value of Alignment Condition
SDATA	SDATA	Allocates a memory range (up to 64 Kbytes, combined with SBSS section), referenced with 1 instructions using gp and 16-bit displacement, having an initial value.	4
SBSS	SBSS	Allocates a memory range (up to 64 Kbytes, combined with SDATA section), referenced with 1 instructions using gp and 16-bit displacement, not having an initial value.	4
SDATA23	SDATA23	Allocates a memory range (up to 8MKbytes, combined with SBSS23 section), referenced with 1 instructions using gp and 23-bit displacement, having an initial value.	4
SBSS23	SBSS23	Allocates a memory range (up to 8Mbytes, combined with SDATA23 section), referenced with 1 instructions using gp and 23-bit displacement, not having an initial value.	4
TDATA	TDATA	Allocates a memory range (up to 256 bytes, in the positive direction from ep), referenced with 1 instructions using ep, having an initial value.	4
TDATA4	TDATA4	Allocates a memory range(up to 16 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 4-bit displacement, having an initial value.	4
TBSS4	TBSS4	Allocates a memory range (up to 16 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 4-bit displacement, not having an initial value.	4

Relocation Attribute	Description Format	Explanation	Default Value of Alignment Condition
TDATA5	TDATA5	Allocates a memory range(up to 32 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 5-bit displacement, having an initial value.	4
TBSS5	TBSS5	Allocates a memory range (up to 32 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 5-bit displacement, not having an initial value.	4
TDATA7	TDATA7	Allocates a memory range(up to 128 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 7-bit displacement, having an initial value.	4
TBSS7	TBSS7	Allocates a memory range (up to 128 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 7-bit displacement, not having an initial value.	4
TDATA8	TDATA8	Allocates a memory range(up to 256 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 8-bit displacement, having an initial value.	4
TBSS8	TBSS8	Allocates a memory range (up to 256 bytes, in the positive direction from ep), referenced with 1 instructions using ep and 8-bit displacement, not having an initial value.	4
EDATA	EDATA	Allocates a memory range (up to 64 Kbytes, combined with EBSS section), referenced with 1 instructions using ep and 16-bit displacement, having an initial value.	4
EBSS	EBSS	Allocates a memory range (up to 64 Kbytes, combined with EDATA section), referenced with 1 instructions using ep and 16-bit displacement, not having an initial value.	4
EDATA23	EDATA23	Allocates a memory range (up to 8 Mbytes, combined with EBSS23 section), referenced with 1 instructions using ep and 16-bit displacement, having an initial value.	4
EBSS23	EBSS23	Allocates a memory range (up to 8 Mbytes, combined with EDATA23 section), referenced with 1 instructions using ep and 16-bit displacement, not having an initial value.	4
ZDATA	ZDATA	Allocates a memory range (up to 32 Kbytes, combined with ZBSS section, in the negative direction from r0), referenced with 1 instructions using r0 and 16-bit displacement, having an initial value.	4
ZBSS	ZBSS	Allocates a memory range (up to 32 Kbytes, combined with ZDATA section, in the negative direction from r0), referenced with 1 instructions using r0 and 16-bit displacement, not having an initial value.	4
ZDATA23	ZDATA23	Allocates a memory range (up to 4 Mbytes, combined with ZBSS23 section, in the negative direction from r0), referenced with 1 instructions using r0 and 23-bit displacement, having an initial value.	4
ZBSS23	ZBSS23	Allocates a memory range (up to 4 Mbytes, combined with ZDATA23 section, in the negative direction from r0), referenced with 1 instructions using r0 and 23-bit displacement, not having an initial value.	4

Relocation Attribute	Description Format	Explanation	Default Value of Alignment Condition
DATA	DATA	Allocates a memory range (up to 4 Gbytes, combined with BSS section, in the negative direction from r0), referenced with 1 instructions using r0 and 32-bit displacement, having an initial value.	4
BSS	BSS	Allocates a memory range (up to 4 Gbytes, combined with DATA section, in the negative direction from r0), referenced with 1 instructions using r0 and 32-bit displacement, not having an initial value.	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.

- The assembler will output an error if a relocation attribute other than "Table 5.8 Relocation Attributes of .dseg" is specified.
- Machine language instructions cannot be described in a data section with BSS relocation attributes. If described, an error is output.
- 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.

Relocation Atribute	Default Section Name
SDATA	.sdata
SBSS	.sbss
SDATA23	.sdata23
SBSS23	.sbss23
TDATA[Note]	.tdata
TDATA4	.tdata4
TBSS4	.tbss4
TDATA5	.tdata5
TBSS5	.tbss5
TDATA7	.tdata7
TBSS7	.tbss7
TDATA8	.tdata8
TBSS8	.tbss8
EDATA	.edata
EBSS	.ebss
EDATA23	.edata23
EBSS23	.ebss23
ZDATA	.zdata
ZBSS	.zbss
ZDATA23	.zdata23

Relocation Atribute	Default Section Name
ZBSS23	.zbss23
DATA	.data
BSS	.bss

Note A specification possible section name is only a default section name in TDATA relocation attributes. But, it is possible to omit.

The default section names have the relocation attributes shown above. Giving them any other attributes is not possible.

- If you wish to specify a number as first character with a section name, enclose each file name with a double quotation (").

  Example

```
test        .dseg   data
            .dw     0x1234
            .dw     0x5678
"test"      .dseg   data
            .dw     0x1234
            .dw     0x5678
"123test"   .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*	[; *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.

.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.
- A .org directive is not allowed for a section that has the TDATA relocation attribute. Doing so will lead to an error and cause processing to end.

.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 (").
  If you wish to specify a number as first character with a section name, enclose each file name with a double quotation (").

  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.

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 5.9 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, the assembler outputs the following message then stops assembling.

  E0550203: illegal expression (*string*)

- If a label name, a macro name defined by the .macro directive, or a symbol of the same name as a formal parameter of a macro is specified, the assembler outputs the following message and stops assembling.

  E0550212: symbol already define as *string*

.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 5.10 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

.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]

- The file name is written with the specified image.
- 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.

.stack

Defines the stack amount of consumption for a symbol.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	.stack	*symble-name=value*	[; *comment*]

[Function]

- The ".stack" directive is compiler debugging information.

[Description]

- This defines the stack amount of consumption for a symbol.
- The stack amount of consumption for a symbol can only be defined once, and subsequent definitions are ignored.
- The stack amount of consumption can only be defined as a 4-byte range of 0x0 to 0xFFFFFFFC. If a different value is specified, then the definition is ignored.
- This is the function information of the C source program that the compiler outputs.

._line_top

Information specified by the compiler #pragma inline_asm statement.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	._line_top	inline_asm	[; *comment*]

[Function]

- The ._line_top directive is the information specified by the compiler #pragma inline_asm statement.

[Description]

- This is the #pragma inline_asm statement information of the C source program that the compiler outputs.
- The ._line_top directive indicates the start of the instructions for a function which has been specified as inline_asm.

[Caution]

- Assembler control instructions are not usable in assembly code for functions specified as inline_asm. In addition, only the directives listed below are usable. Specifying any other directive will lead to an error.
  - data definition directives (.db/.db2/.dhw/.db4/.dw/.db8/.ddw/.dshw/.ds/.float/.double)
  - macro directives (.macro/.irp/.rept/.local)
- Defining labels other than local labels is forbidden in functions specified as inline_asm. Defining any other labels will lead to errors.

._line_end

Information specified by the compiler #pragma inline_asm statement.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
	._line_end	inline_asm	[; *comment*]

[Function]

- The ._line_end directive is the information specified by the compiler #pragma inline_asm statement.

[Description]

- This is the #pragma inline_asm statement information of the C source program that the compiler outputs.
- The ._line_end directive indicates the end of the instructions for a function which has been specified as inline_asm.

[Caution]

- Assembler control instructions are not usable in assembly code for functions specified as inline_asm. In addition, only the directives listed below are usable. Specifying any other directive will lead to an error.
  - data definition directives (.db/.db2/.dhw/.db4/.dw/.db8/.ddw/.dshw/.ds/.float/.double)
  - macro directives (.macro/.irp/.rept/.local)
- Defining labels other than local labels is forbidden in functions specified as inline_asm. Defining any other labels will lead to errors.

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 secures the area for memory used by the program.
The following data definition and partitioning directives are available.

Table 5.11 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	Secures 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	(*absolute-expression*)	[; *comment*]
[*label*:]	.db	*expression*[, ...]	[; *comment*]
[*label*:]	.db	*"Character string constants"*	[; *comment*]

[Function]

- The .db directive tells the assembler to initialize a byte area.
  The number of bytes to be initialized can be specified as "size".
- The .db directive also 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]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.

  (a) With size specification:

  <1> If a size is specified in the operand field, the assembler initializes an area equivalent to the specified number of bytes with the value "0".

  <2> An absolute expression can be described as a size. If the size description is illegal, the CC-RH outputs an error message and will not execute initialization.

  (b) With initial value specification:

  <1> 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.

  <2> Character string constants
  If the first operand is surrounded by corresponding double quotes ("), then it is assumed to be a string constant.
  If a character string constants is described as the operand, an 8-bit ASCII code 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]

```
                .cseg   text
WORK1:          .db     (1)                     ; (1)
WORK2:          .db     (2)                     ; (1)
                .cseg   text
MASSAG:         .db     "ABCDEF"                ; (2)
DATA1:          .db     0xA, 0xB, 0xC           ; (3)
DATA2:          .db     (3 + 1)                 ; (4)
DATA3:          .db     "AB" + 1                ; (5)   <- Error
```

(1) Because the size is specified, the assembler will initialize each byte area with the value "0".

(2) A 6-byte area is initialized with character string 'ABCDEF'

(3) A 3-byte area is initialized with "0xA, 0xB, 0xC".

(4) A 4-byte area is initialized with "0x0".

(5) This description occurs in an error.

.db2/.dhw

Initialization of 2-byte area.

[Syntax]

Symbol field	Mnemonic field	Operand field	Comment field
[*label*:]	.db2	(*absolute-expression*)	[; *comment*]
[*label*:]	.db2	*expression*[, ...]	[; *comment*]
[*label*:]	.dhw	(*absolute-expression*)	[; *comment*]
[*label*:]	.dhw	*expression*[, ...]	[; *comment*]

[Function]

- The .db2 and .dhw directive tells the assembler to initialize 2-byte area.
  The number of 2-byte data to be initialized can be specified as "size".
- The .db2 and .dhw directive also 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]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.

  (a) With size specification:

  <1> If a size is specified in the operand field, the assembler initializes an area equivalent to the specified number of 2-byte with the value "0".

  <2> An absolute expression can be described as a size. If the size description is illegal, the CC-RH outputs an error message and will not execute initialization.

  (b) With initial value specification:

  <1> 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 secured 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 secured.
- 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	(*absolute-expression*)	[; *comment*]
[*label*:]	.db4	*expression*[, ...]	[; *comment*]
[*label*:]	.dw	(*absolute-expression*)	[; *comment*]
[*label*:]	.dw	*expression*[, ...]	[; *comment*]

[Function]

- The .db4 and .dw directive tells the assembler to initialize 4-byte area.
  The number of 4-byte data to be initialized can be specified as "size".
- The .db4 and .dw directive also 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]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.

  (a) With size specification:

  <1> If a size is specified in the operand field, the assembler initializes an area equivalent to the specified number of 4-byte with the value "0".

  <2> An absolute expression can be described as a size. If the size description is illegal, the CC-RH outputs an error message and will not execute initialization.

  (b) With initial value specification:

  <1> 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*:]	.db8	*absolute-expression*[, ...]	[; *comment*]
[*label*:]	.ddw	(*absolute-expression*)	[; *comment*]
[*label*:]	.ddw	*absolute-expression*[, ...]	[; *comment*]

[Function]

- The .db8 and .ddw directive tells the assembler to initialize 8-byte area.
  The number of 8-byte data to be initialized can be specified as "size".
- The .db8 and .ddw directive also 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]

- If a value in the operand field is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.

  (a) With size specification:

  <1> If a size is specified in the operand field, the assembler initializes an area equivalent to the specified number of 8-byte with the value "0".

  <2> An absolute expression can be described as a size. If the size description is illegal, the CC-RH outputs an error message and will not execute initialization.

  (b) With initial value specification:

  <1> 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 secured as a single-precision floating-point number. Consequently, the value of the expression must be between 1.40129846e-45 and 3.40282347e+3. In other cases, the data from the lower four bytes will be secured as a single-precision floating-point number.
- 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 secured as a double-precision floating-point number. Consequently, the value of the expression must be between 4.9406564584124654e-324 and 1.7976931348623157e+308. In other cases, the data from the lower eight bytes will be secured as a double-precision floating-point number.
- 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

Secures 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*]
[*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 a value in the first operand is parenthesized, the assembler assumes that a size is specified. Otherwise, an initial value is assumed.
- The first operand is a size specification. If a second operand is also specified, then it will be treated as the initial value for that value.

  (a) With size specification:

  <1> If a size is specified in the operand, then if an initial value is specified, the compiler will fill the specified number of bytes with the specified value; otherwise, it will fill that number of bytes with zeroes ("0"). Note, however, that no area will be secured if the specified number of bytes is 0.

  <2> An absolute expression can be described as a size. If the size description is illegal, the CC-RH outputs an error message and will not execute initialization.

  (b) With initial value specification:

  <1> Expression
  The value of an expression must be byte data. Therefore, the value of the operand must be in the range of 0x0 to 0xFF. If the value exceeds byte, the assembler will use only lower 1-byte of the value as valid data.

- 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 this directive is "BSS", then an error is output and this directive is ignored because initial values cannot be specified.

.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.

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 5.12 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 (other than bit symbols) 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) Externally referenced symbol (symbol defined with the .extern directive)

  (c) Section name

  (d) Module name

  (e) Macro name

  (f) Undefined symbol in the source program

[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 (other than bit symbols) 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]

- When referencing a symbol defined in another module, the .extern directive must be used to declare the symbol as an external reference.

[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.
- An error will occur if the symbol specified as the first operand has already been defined or declared with a .public or .extern directive in the given module. The symbol name is output in the error message.
- Names of macros cannot be used as the first operand.

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 5.13 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, the CC-RH outputs the message.
- For the macro name to be described in the symbol field, see the conventions of symbol description in "(2) Symbol".
- To reference a macro, describe the defined macro name in the mnemonic field.
- 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.
- An error occurs if any reserved word is described as a formal parameter. However, if a user-defined symbol is described, its recognition as a formal parameter will take precedence.
- The number of formal parameters must be the same as the number of actual parameters. If a shortage of actual parameters, the CC-RH outputs the error message.
- The theoretical maximum number of formal parameters is 4,294,967,294 (i.e. 0xFFFFFFFE). The actual number that can be used depends on the amount of memory, however.
- A name or label defined within the macro body if declared with the .local directive becomes effective with respect to one-time macro expansion.
- 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.
- Formal parameter definition lines, reference lines, and symbol names are not output to a cross-reference list.
- Two or more sections must not be defined in a macro body. If defined, an error will be output.
- An error will be output if there are extra formal parameters that are not referenced in the macro body.
- If an undefined macro is called in a macro body, the CC-RH outputs the message then stops assembling.
- If a currently defined macro is called in a macro body, the CC-RH outputs the message then stops assembling.
- If a parameter defined by a label or directive is specified for a formal parameter, the CC-RH outputs the message and stops assembling.

- 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), addressing-method specification label (#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. Such as operand can't designate the part of the sentence. If operand has a macro call, performs a label reference is undefined macro name, or the CC-RH outputs the message then stops assembling.
- 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 theoretical maximum number of symbol names is 4,294,967,294 (i.e. 0xFFFFFFFE). The actual number that can be used depends on the amount of memory, however.
- Specifying 4,294,967,294 or more local symbols as formal parameters to ".local" quasi directives will cause the following error message to be output, and the assembly will halt.

  F0550514 : Paramater table overflow.

- Local symbol names generated by the assembler are generated in the range of .??00000000 to .??FFFFFFFF.
- Only an undefined symbol or a symbol that has been declared as a local symbol can be declared as a local symbol.
- A specific symbol that will replace the specified symbol name is generated for each declaration.
- To make the declared symbol valid, the symbol should be defined as a label.
- The symbol name declared as a local symbol cannot be defined for any purpose other than a label.

[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 by the assembler is terminated.
- Assembly control instructions may be described in the REPT-ENDM block.
- Macro definitions cannot be described in the REPT-ENDM block.
- The value is evaluated as a 32-bit signed integer.
- If there is no arrangement of statements (block), nothing is executed.
- If the result of evaluating the expression is negative, the CC-RH outputs the message then stops assembling.
- An error will be output if a macro is defined in the macro body of a macro definition, and processing will continue, without performing expansion.

[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 CC-RH 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 theoretical maximum number of actual parameters is 4,294,967,294 (i.e. 0xFFFFFFFE). The actual number that can be used depends on the amount of memory, however.
- If the same parameter name is specified for a formal parameter and an actual parameter, the CC-RH outputs the message and stops assembling.
- If a parameter defined by a label or other directive is specified for a formal parameter and an actual parameter, the CC-RH outputs the message and stops assembling.
- An error will be output if a macro is defined in the macro body of a macro definition, and processing will continue, without performing expansion.

[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 CC-RH 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 CC-RH 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 CC-RH outputs the message then stops assembling.

[Example]

(1) MACRO-ENDM

```
ADMAC   .macro  PARA1, PARA2
                mov     A, #PARA1
                add     A, #PARA2
        .endm
```

(2) REPT-ENDM

```
.cseg   text
  :
.rept   3
        inc     B
        DEC     C
.endm
```

(3) IRP-ENDM

```
.cseg   text
  :
.irp    PARA, <1, 2, 3>
        add     A, #PARA
        mov     [DE], A
.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 5.14 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 5.15 Assembler Control Instructions

Control Instruction	Overview
REG_MODE	Outputs a register mode information section
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 to the data or bss 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 section is output.

[Syntax]

```
[Δ]$[Δ]REG_MODE[Δ]specify-register-mode[Δ][;comment]
```

[Function]

- A register mode information section 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 section 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 register mode 22 is used, then there are 5 working registers and 5 register-variable registers; and if register mode 32 is used, then there are 10 of each.
- If register mode 32 is used, a register mode information section is not output into the object module file generated by the assembler.
- If the register mode of this control instruction differs from the register mode specified via options, then CC-RH 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. CC-RH 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

Assumes that external data having symbol name extern_symbol has been allocated to the data or bss attribute section, and expands the instructions which reference that data.

[Syntax]

```
[Δ]$[Δ]DATA[Δ]symbol-name[Δ][;comment]
```

[Function]

- Assumes that external data having symbol name extern_symbol has been allocated to the data or bss attribute section, and expands the instructions which reference that data.
- This format is used when a variable for which "data" is specified in #pragma section is externally referenced by an assembler source file.

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]symbol-name[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.
- This format is used when a variable for which "sdata" is specified in #pragma section is externally referenced by an assembler source file.

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 CC-RH can input an assembler source file or binary file to a specified position. The following file input control instructions are available.

Table 5.16 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 CC-RH 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 will be output, and assembly will continue assuming the include file is closed.
- Section definition directive, macro definition directives, and conditional assembly control instructions that are not targets for assembly are not checked.

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 CC-RH 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 5.17 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]

- This control instruction can be placed in an ordinary source program.
- 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 or cross-reference 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]

- This control instruction can be placed in an ordinary source program.
- 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 or cross-reference 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.

[Description]

- This control instruction can be placed in an ordinary source program.
- Absolute expressions are evaluated as 32-bit signed integers.

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.

[Description]

- This control instruction can be placed in an ordinary source program.
- Absolute expressions are evaluated as 32-bit signed integers.

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.

[Description]

- This control instruction can be placed in an ordinary source program.
- Absolute expressions are evaluated as 32-bit signed integers.

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.

[Description]

- This control instruction can be placed in an ordinary source program.
- Absolute expressions are evaluated as 32-bit signed integers.

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.

[Description]

- This control instruction can be placed in an ordinary source program.

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.

[Description]

- This control instruction can be placed in an ordinary source program.

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 "~" concatenates one character or one character string to another within a macro body. At macro expansion time, the character or character string on the left of the concatenation is concatenated to the character or character string on the right of the sign. The "~" itself disappears after concatenating the strings.
- The symbols before and after the combination symbol "~" in the symbols of a macro definition can be recognized as formal parameters or local symbols, and combination symbols can also be used as delimiter symbols. At macro expansion time, strings before and after the "~" in the symbol are evaluated as the local symbols and formal parameters, and concatenated into single symbols.
- 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.
- Two "~" signs in succession are handled as a single "~" sign.

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
```

(2) $ (Dollar symbol)

If a symbol prefixed with a dollar symbol ($) is specified as an actual argument for a macro call, the assembler assumes the symbol to be specified as an actual argument. If, however, an identifier other than a symbol or an undefined symbol name is specified immediately after the dollar symbol ($), the as850 outputs the message then stops assembling.

Example

```
mac1    .macro  x
        mov     x, r10
.endm
mac2    .macro
value   .set    10
        mac1    value
        mac1    $value
.endm
mac2
```

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
- 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.

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 5.18 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 5.19 Meaning of Symbols

Symbols	Meaning
CMD	Instruction
CMDi	Instruction(andi, ori, or 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	Offset reference of label within the section (no instruction expansion) or offset reference of ep
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)
vector	Trap vector (0 to 31)
BITIO	Peripheral I/O register (for 1-bit manipulation only)

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. When r0 is specified as a destination register, the assembler outputs the following message[Note], and 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, r0
    ↓
W0550013: register r1 used as destination register.
```

(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: register r1 used as destination register.
```

```
mov     r1, r10
    ↓
W0550013: register r1 used as source register.
```

(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 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 5.20 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 or mov 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 secured.
!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 time of link.
	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 secured.
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 secured.
%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 time of link.
	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 secured.

Reference Method	Instructions Used	Meaning
$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 5.21 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 time of link. 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 time of link. 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 time of link. Part of [reg] can be omitted. If omitted, the assembler assumes that [ep] has been specified.

Note See "(h) gp offset reference".

Table 5.22 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.
$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 time of link.
!label	The absolute address of label (*label*) is treated as an immediate. This has a 16-bit value. If operation instruction of an architecture for which a 16-bit value can be specify Note 2 as an immediate are specified, and instruction is not expanded. If the add, mov, or mulh instruction is specified, expansion into appropriate one instruction is possible. No other instructions can be specified. If the value is not within a range that can be expressed in 16 bits, an error is output at the time of link.
%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 of an architecture for which a 16-bit value can be specify Note 2 as an immediate are specified, and instruction is not expanded. This reference method can be specified only for operation instructions of an architecture for which a 16-bit value can be specified as an immediate, and add, mov, and mulh instructions. If the add, mov, or mulh instruction is specified, expansion into appropriate one instruction is possible. No other instructions can be specified. If the value is not within a range that can be expressed in 16 bits, an error is output at the time of link.

Note 1. See "(h) gp offset reference".

Note 2. The instructions for which a 16-bit value can be specified as an immediate are the addi, andi, movea, mulhi, ori, satsubi, and xori instructions.

Table 5.23 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 time of link.

Table 5.24 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

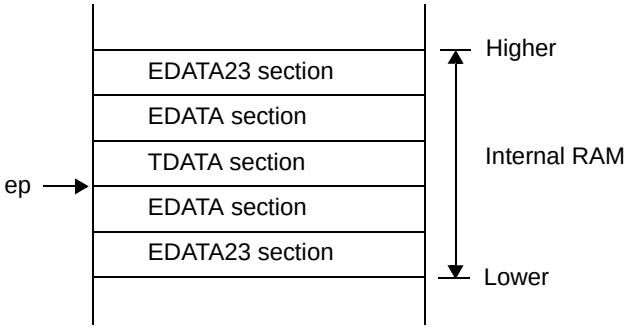
This section describes the ep offset reference. The CC-RH assumes that data explicitly stored in internal RAM is shown below.

Reference through the offset from address indicated by the element pointer (ep).

Data in the internal RAM is divided into the following two groups.

- TDATA/TDATA4/TBSS4/TDATA5/TBSS5/TDATA7/TBSS7/TDATA8/TBSS8 section (Data is referred by memory reference instructions (sld/sst) in a small code size)
- EDATA/EBSS section (Data is referred by memory reference instructions (ld/st) in a large code size)
- EDATA23/EBSS23 section (Data is referred by memory reference instructions (ld23/st23) in a large code size)

Figure 5.2 Memory Location Image of Internal RAM



<1> Data allocation

In internal RAM, data is allocated to the sections as follows:

- When developing a program in C
  Allocate data by specifying the "sdata", or "sdata23" section type 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, sdata, sbss, sdata23, or sbss23 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.

Example

```
        .dseg   DATA
label:
        .dhw    0x0001
        .cseg   TEXT
        ld.h    %label[ep], r20
```

Instructions are coded to allocate a label to the EDATA section and to perform reference by ep offset. However, label is allocated to the DATA section because of the allocation error. In this case, the assembler loads the data in the base register ep symbol value + offset value in the DATA section of label.

(h) gp offset reference

This section describes the gp offset reference. The CC-RH assumes that data stored in external RAM (other than .sedata/.sebss section explained on the previous page) is basically shown below.

Referred by the offset from the address indicated by global pointer (gp).

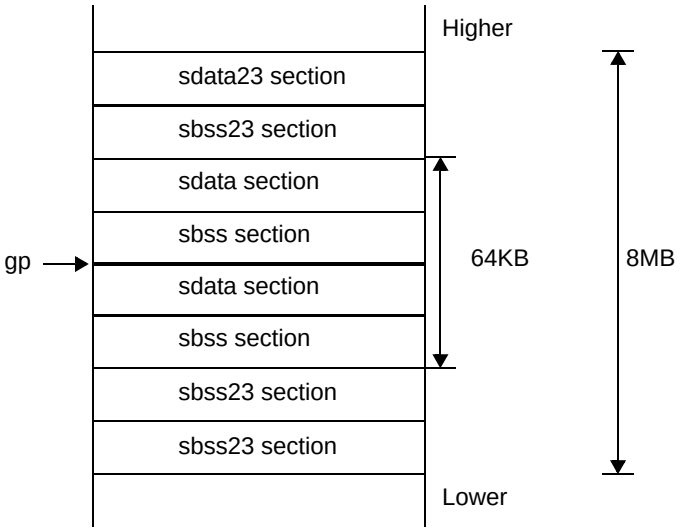
If r0-relative memory allocation for internal ROM or RAM is not done with the "#pragma section" command of C, or an assembly language section definition directive, all data is subject to gp offset reference.

<1> Data allocation

The memory reference instruction (ld/st) of the machine instruction of the RH850 family can accept 16-bit immediate or 23-bit immediate as a displacement. For this reason, the CC-RH classifies data into the following three types. Data of the first type is allocated to the sdata or sbss section, the second type is allocated to the sdata23 or sbss23 section, the 3rd type is allocated to the data or bss section.

- Data allocated to a memory range that can be referred by using the global pointer (gp) and a 16-bit displacement.
- Data allocated to a memory range that can be referred by using the global pointer (gp) and a 23-bit displacement.
- Data allocated to a memory range that can be referred by using the global pointer (gp) and (constructed by many instructions) a 32-bit displacement.

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.

Data in the sdata/sbss/sdata23/sbss23 sections can be referred by using a single instruction. To reference data in the data/bss sections, however, two or more instructions are necessary. Therefore, the more data allocated to the sdata/sbss/sdata23/sbss23 sections, the higher the execution efficiency and object efficiency of the generated machine instructions. However, the size of the memory range that can be referred with a 16-bit and 23-bit displacement is limited.
If all the data cannot be allocated to the sdata/sbss/sdata23/sbss23 sections, it becomes necessary to determine which data is to be allocated to the sdata/sbss/sdata23/sbss23 sections.
The CC-RH "allocates as much data as possible to the sdata/sbss/sdata23/sbss23 sections". By default, all data items are allocated to the sdata/sbss/sdata23/sbss23 sections. The data to be allocated can be selected as follows:

- When using a program to specify the section to which data will be allocated.
  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
Using the data allocation method explained above, the assembler generates a machine instruction string that performs:

- Reference by using a 16-bit displacement for gp offset reference to data allocated to the sdata- and sbss-attribute sections.
- Reference by using a 32-bit displacement (consisting of two or more machine instructions) for gp offset reference to data allocated to the data- and bss-attribute sections.

Example

```
        .dseg   DATA
data:   .db4    0xFFF00010      ; (1)
        .cseg   TEXT
        ld.w    $data[gp], r20  ; (2)
```

The assembler generates a machine instruction string, equivalent to the following instruction string for the ld.w instruction in (2), that performs gp offset reference of the data defined in (1).[Note]

```
movhi   HIGHW1($data), gp, r1
ld.w    LOWW($data)[r1], r20
```

Note See "(i) About HIGH/LOW/HIGHW/LOWW/HIGHW1", for details of HIGHW1/LOWW.

The assembler processes files on a one-by-one basis. Consequently, it can identify to which attribute section data having a definition in a specified file has been allocated, but cannot identify the section to which data not having a definition in a specified file has been allocated.
To develop a program in an assembly language, therefore, specify the size of the data (actually, a label for which there is no definition in a specified file and which is referred by a gp offset) for which there is no definition in a specified file, by using the .extern directives.

```
.extern data, 4          ; (1)
.cseg   TEXT
ld.w    $data[gp], r20   ; (2)
```

To develop a program in C, the C compiler of the CC-RH automatically generates the .extern directive, thus output the code which specifies the size of data not having a definition in the specified file (actually, a label for which there is no definition in a specified file and which is referred by a gp offset).

Remark The handling of gp offset reference (specifically, memory reference instructions that use a relative expression having the gp offset of a label as their displacement) by the assembler is summarized below.

- 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.

  Note If the value of the constant expression of a relative expression in the form of "label + constant expression" exceeds 16 bits, the assembler generates a machine instruction string that performs reference 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 store 32-bit constant value in a register
To store a 32-bit constant value in a register, the assembler performs instruction expansion, and generates an instruction string, by using the movhi and movea instructions. These divide the 32-bit constant value into the higher 16 bits and lower 16 bits.

Example

`mov  0x18000, r11`	`movhi  HIGHW1(0x18000), r0, r1` `movea  LOWW(0x18000), r1, r11`

At this time, the movea instruction, used to store the lower 16 bits in the register, sign-extends the specified 16-bit value to a 32-bit value. To adjust the sign-extended bits, the assembler does not merely store the higher 16 bits in a register when using the movhi instruction, instead it stores the following value in the register.

Higher 16 bits + the most significant bit (bit of bit number 15) of the lower 16 bits

<2> 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

<3> 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 5.25 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
            :
        movhi   HIGHW1 ($L1), r0, r1  ; Stores the gp offset value of L1 in r10
        movea   LOWW ($L1), r1, r10
```

5.9 Description of Instructions

This section describes the instructions of the assembly language supported by the assembler.
For details of the machine instructions generated by the assembler, see the "Each Device User Manual".

Instruction

Indicates the meaning of instruction.

[Syntax]

Indicates the syntax of instruction.

[Function]

Indicates the function of instruction.

[Description]

Indicates the operating method of instruction.

[Flag]

Indicates the operation of flag (PSW) by the execution of instruction.
However, in (set1, clr1, not1) bit operation instruction, indicates the flag value before execution.
"---" of table indicates that the flag value is not changed.

[Caution]

Indicates the caution in instruction.

5.9.1 Load/Store instructions

This section describes the load/store instructions. Next table lists the instructions described in this section.
See the RH850 product user's manual and architecture edition for details.

Table 5.26 Load/Store Instructions

Instruction		Meaning
ld	ld.b	Byte data load
	ld.h	Halfword data load
	ld.w	Word data load
	ld.bu	Unsigned byte data load
	ld.hu	Unsigned halfword data load
sld	sld.b	Byte data load (short format)
	sld.h	Halfword data load (short format)
	sld.w	Word data load (short format)
	sld.bu	Unsigned byte data load (short format)
	sld.hu	Unsigned halfword data load (short format)
ld23	ld23.b	Byte data load
	ld23.h	Halfword data load
	ld23.w	Word data load
	ld23.bu	Unsigned byte data load
	ld23.hu	Unsigned halfword data load
	ld23.dw	Doubleword data load
st	st.b	Byte data store
	st.h	Halfword data store
	st.w	Word data store
sst	sst.b	Byte data store (short format)
	sst.h	Halfword data store (short format)
	sst.w	Word data store (short format)
st23	st23.b	Byte data store
	st23.h	Halfword data store
	st23.w	Word data store
	st23.dw	Doubleword data store

ld

Data is loaded.

[Syntax]

- ld.b disp[reg1], reg2
- ld.h disp[reg1], reg2
- ld.w disp[reg1], reg2
- ld.bu disp[reg1], reg2
- ld.hu disp[reg1], reg2

The following can be specified for displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with LOWW applied

[Function]

The ld.b, ld.bu, ld.h, ld.hu, and ld.w instructions load data of 1 byte, 1 halfword, and 1 word, from the address specified by the first operand, int the register specified by the second operand.

[Description]

- If any of the following is specified for disp, the assembler generates one ld machine instruction[Note]. In the following explanations, ld denotes the ld.b/ld.h/ld.w/ld.bu/ld.hu instructions.

(a) Absolute expression having a value in the range of -32,768 to +32,767

`ld disp16[reg1], reg2`	`ld disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`ld $label[reg1], reg2`	`ld $label[reg1], reg2`

(c) Absolute expression having !label or %label

`ld !label[reg1], reg2`	`ld !label[reg1], reg2`
`ld %label[reg1], reg2`	`ld %label[reg1], reg2`

(d) Expression with , LOWW

`ld disp16[reg1], reg2`	`ld disp16[reg1], reg2`

Note The ld machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement

- If any of the following is specified for disp, 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

`ld disp[reg1], reg2`	`movhi HIGHW1(disp), reg1, r1` `ld LOWW(disp)[r1], reg2`

(b) Relative expression having #label or label, or that having $label for a label having definition in the sdata/sbss-attribute section

`ld #label[reg1], reg2`	`movhi HIGHW1(#label), reg1, r1` `ld LOWW(#label)[r1], reg2`
`ld label[reg1], reg2`	`movhi HIGHW1(label), reg1, r1` `ld LOWW(label)[r1], reg2`
`ld $label[reg1], reg2`	`movhi HIGHW1($label), reg1, r1` `ld LOWW($label)[r1], reg2`

- If disp is omitted, the assembler assumes 0.
- 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 [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.
- Specify an ld23 instruction to specify an ld instruction with a 23 bit-wide disp..

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- ld.b and ld.h sign-extend the data of 1 byte and 1 halfword, respectively, and load the data into a register as 1 word.
- If a value that is not a multiple of 2 is specified as disp of ld.h, ld.w, or ld.hu, the assembler and link editor aligns disp with 2 and generates a code. Then, the assembler and link editor outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

- If r0 is specified as the second operand of ld.bu and ld.hu, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

sld

Data is loaded (short format).

[Syntax]

- sld.b disp7[ep], reg2
- sld.h disp8[ep], reg2
- sld.w disp8[ep], reg2
- sld.bu disp4[ep], reg2
- sld.hu disp5[ep], reg2

The following can be specified for displacement (disp4/5/7/8):

- Absolute expression having a value of up to 7 bits for sld.b, 8 bits for sld.h and sld.w, 4 bits for sld.bu, and 5 bits for sld.hu.
- Relative expression

[Function]

The sld.b, sld.bu, sld.h, sld.hu, and sld.w instructions load the data of 1 byte, 1 halfword, and 1 word, from the address obtained by adding the displacement specified by the first operand to the contents of register ep, to the register specified by the second operand.

[Description]

The assembler generates one sld machine instruction. Base register specification "[ep]" can be omitted.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- sld.b and sld.h sign-extend and store data of 1 byte and 1 halfword, respectively, in the register as 1 word.
- If a value that is not a multiple of 2 is specified as disp8 of sld.h or disp5 of sld.hu, and if a value that is not a multiple of 4 is specified as disp8 of sld.w, the assembler aligns disp8 or disp5 with multiples of 2 and 4, respectively, and generates a code. Then, the assembler and link editor outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

- If a value exceeding 127 is specified for disp7 of sld.b, a value exceeding 255 is specified for disp8 of sld.h and sld.w, a value exceeding 16 is specified for disp4 of sld.bu, and a value exceeding 32 is specified for disp5 of sld.hu, the assembler outputs the following message, and generates code in which disp7, disp8, disp4, and disp5 are masked with 0x7F, 0xFF, 0xF, and 0x1F, respectively.

```
W0550011 : Illegal operand (range error in immediate).
```

- If r0 is specified as the second operand of the sld.bu and sld.hu, the assembler outputs the following message and stops assembling.

```
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

ld23

Data is loaded.

[Syntax]

- ld23.b disp23[reg1], reg2
- ld23.h disp23[reg1], reg2
- ld23.w disp23[reg1], reg2
- ld23.bu disp23[reg1], reg2
- ld23.hu disp23[reg1], reg2
- ld23.dw disp23[reg1], reg2

The following can be specified for displacement (disp):

- Absolute expression having a value of up to 23 bits
- Relative expression

[Function]

The ld23.b, ld23.bu, ld23.h, ld23.hu, and ld23.w instructions load data of 1 byte, 1 halfword, and 1 word, from the address specified by the first operand, int the register specified by the second operand. The ld23.dw instruction loads a double word of data from the address specified in the first operand, then register reg2 specified in the second operand into the lower 32 bits, and reg2 + 1 into the upper 32 bits.

[Description]

- The assembler generates a 48-bit ld[Note] instruction in machine language.

Note The ld machine instruction takes an immediate value in the range of -4,194,304 to +4,194,303 (0xFFC00000 to 0x3FFFFF) as the displacement

- If disp23 is omitted, the assembler assumes 0.
- If a relative expression having #labelis specified as disp23, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression having $label is specified as disp23, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- ld23.b and ld23.h sign-extend the data of 1 byte and 1 halfword, respectively, and load the data into a register as 1 word.
- If a value that is not a multiple of 2 is specified as disp of ld23.h, ld23.w, ld23.hu, or ld23.dw, the assembler and link editor aligns disp with 2 and generates a code. Then, the assembler and link editor outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

- If r0 is specified as the second operand of ld.bu and ld.hu, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in V850E mode).

- A message is output if an odd-numbered register is specified in the second operand of the ld23.dw instruction.

W0550013 : *register* used as *kind* register.

st

Data is stored.

[Syntax]

- st.b reg2, disp[reg1]
- st.h reg2, disp[reg1]
- st.w reg2, disp[reg1]

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with LOWW applied

[Function]

The st.b, st.h, and st.w instructions store the data of the lower 1 byte, lower 1 halfword, and 1 word, respectively, of the register specified by the first operand to the address specified by the second operand.

[Description]

- If any of the following is specified as disp, the assembler generates one st machine instruction[Note]. In the following explanations, st denotes the st.b/st.h/st.w instructions.

(a) Absolute expression having a value in the range of -32,768 to +32,767

`st reg2, disp16[reg1]`	`st disp16[reg1], reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`st reg2, $label[reg1]`	`st $label[reg1], reg2`

(c) Absolute expression having !label or %label

`st reg2, !label[reg1]`	`st !label[reg1], reg2`
`st reg2, %label[reg1]`	`st %label[reg1], reg2`

(d) Expression withLOWW

`st reg2, disp16[reg1]`	`st disp16[reg1], reg2`

Note The st machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement.

- If any of the following is specified as disp, the assembler executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`st reg2, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `st LOWW(disp)[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

`st reg2, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `st LOWW(#label)[r1], reg2`
`st reg2, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `st LOWW(label)[r1], reg2`
`st reg2, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `st LOWW($label)[r1], reg2`

- 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. If omitted, the assembler assumes that [r0] is 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. If omitted, the assembler assumes that [gp] is specified.
- Specify an st23 instruction to specify an st instruction with a 23 bit-wide disp.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a value that is not a multiple of 2 is specified as the disp of st.h or st.w, the assembler aligns disp with 2 and generates a code. Then, the assembler outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

sst

Data is stored (short format).

[Syntax]

- sst.b reg2, disp7[ep]
- sst.h reg2, disp8[ep]
- sst.w reg2, disp8[ep]

The following can be specified for displacement (disp7/8):

- Absolute expression having a value of up to 7 bits for sst.b or 8 bits for sst.h and sst.w
- Relative expression

[Function]

The sst.b, sst.h, and sst.w instructions store the data of the lower 1 byte, lower 1 halfword, and 1 word, respectively, of the register specified by the first operand to the address obtained by adding the displacement specified by the second operand to the contents of register ep.

[Description]

The assembler generates one sst machine instruction. Base register specification "[ep]" can be omitted.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a value that is not a multiple of 2 is specified as disp8 of sst.h, and if a value that is not a multiple of 4 is specified as disp8 of sst.w, the assembler aligns disp8 with multiples of 2 and 4, respectively, and generates a code. Then, the assembler outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

- If a value exceeding 127 is specified as disp7 of sst.b, and if a value exceeding 255 is specified as disp8 of sst.h and sst.w, the assembler outputs the following message, and generates codes disp7 and disp8, masked with 0x7F and 0xFF, respectively.

W0550011 : Illegal operand (range error in immediate).

st23

Data is stored.

[Syntax]

- st23.b reg2, disp23[reg1]
- st23.h reg2, disp23[reg1]
- st23.w reg2, disp23[reg1]
- st23.dw reg2, disp23[reg1]

The following can be specified as a displacement (disp):

- Absolute expression having a value of up to 23 bits
- Relative expression

[Function]

The st23.b, st23.h, and st23.w instructions store the data of the lower 1 byte, lower 1 halfword, and 1 word, respectively, of the register specified by the first operand to the address specified by the second operand. The st23.dw instruction loads the word data from the register specified in the first operand into the lower 32 bits, and the word data at reg2 + 1 into the upper 32 bits, and then stores this double-word data into the address specified in the second operand.

[Description]

- The assembler generates one st machine instruction[Note].

Note The st machine instruction takes an immediate value in the range of -4,194,304 to +4,194,303 (0xFFC00000 to 0x3FFFFF) as the displacement.

- If disp23 is omitted, the assembler assumes 0.
- If a relative expression with #label is specified as disp23, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression with $label is specified as disp23, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a value that is not a multiple of 2 is specified as the disp of st.h or st.w, the assembler aligns disp with 2 and generates a code. Then, the assembler outputs either one of the following messages.

W0550010 : Illegal displacement in ld instruction.
E0562332 : Relocation value is odd number : "*file*"-"*section*"-"*offset*"

- A message is output if an odd-numbered register is specified in the first operand of the st23.dw instruction.

W0550013 : *register* used as *kind* register.

5.9.2 Arithmetic operation instructions

This section describes the arithmetic operation instructions. Next table lists the instructions described in this section. See the RH850 product user's manual and architecture edition for details.

Table 5.27 Arithmetic Operation Instructions

Instruction	Meaning
add	Adds
addi	Adds (immediate)
adf	Adds with condition
sub	Subtracts
subr	Subtracts reverse
sbf	Subtracts with condition
mulh	Multiplies signed data (halfword)
mulhi	Multiplies signed data (halfword immediate)
mul	Multiplies signed data (word)
mulu	Multiplies unsigned data
mac	Multiplies and adds signed word data
macu	Multiplies and adds unsigned word data
divh	Divides signed data (halfword)
div	Divides signed data (word)
divhu	Divides unsigned data (halfword)
divu	Divides unsigned data (word)
divq	Division of (signed) word data (variable steps)
divqu	Division of (unsigned) word data (variable steps)
cmp	Compares
mov	Moves data
movea	Moves execution address
movhi	Moves higher half-word
mov32	Moves 32-bit data
cmov	Moves data depending on the flag condition
setf	Sets flag condition
sasf	Sets the flag condition after a logical left shift
bins	Insert bit in register
rotl	Rotate

See the device with an instruction set of RH850 product user's manual and architecture edition for details about the device with an instruction set of RH850.

add

Adds.

[Syntax]

- add reg1, reg2
- add imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "add reg1, reg2"
  Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result into the register specified by the second operand.
- Syntax "add imm, reg2"
  Adds the value of the absolute expression or relative expression specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- If this instruction is executed in syntax "add reg1, reg2", the assembler generates one add machine instruction.
- If the following is specified as imm in syntax "add imm, reg2", the assembler generates one add machine instruction-Note.

(a) Absolute expression having a value in the range of -16 to +15

`add imm5, reg`	`add imm5, reg`

Note The add machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the first operand.

- If the following is specified for imm in syntax "add 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

`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

`add imm, reg`	`movhi HIGHW(imm), r0, r1` `add r1, reg`

Else

`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

`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

`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`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

addi

Adds immediate.

[Syntax]

- addi imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

Adds the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied, specified by the first operand, to the value of the register specified by the second operand, and stores the result into the register specified by the third operand.

[Description]

- If the following is specified for imm, the assembler generates one addi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`addi imm16, reg1, reg2`	`addi imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`addi $label, reg1, reg2`	`addi $label, reg1, reg2`

(c) Relative expression having !label or %label

`addi !label, reg1, reg2`	`addi !label, reg1, reg2`
`addi %label, reg1, reg2`	`addi %label, reg1, reg2`

(d) Expression with HIGHW, LOWW, or HIGHW1

`addi imm16, reg1, reg2`	`addi imm16, reg1, reg2`

Note The addi machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF)as the first operand.

- If the following is specified for imm, the assembler executes instruction expansion to generate two or more 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

`addi imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `add reg1, r1`

Else

`addi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `add reg1, reg2`

<2> Else
If reg2 is r0

`addi imm, reg1, r0`	`mov imm, r1` `add reg1, r1`

Else

`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

`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

`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`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

adf

Adds on condition flag.

[Syntax]

- adf imm4, reg1, reg2, reg3
- adf*cnd* reg1, reg2, reg3

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits (0xD cannot be specified)

[Function]

- Syntax "adf imm4, reg1, reg2, reg3"
  It compares the current flag condition with the flag condition indicated by the value of the lower 4 bits of the absolute expression (see "Table 5.28 adfcnd Instruction List") specified by the first operand.
  If the values match, adds the word data of the register specified by the second operand to the word data of the register specified by the third operand. And 1 is added to the addition result and that result is stored in the register specified by the fourth operand.
  If the values not match, adds the word data of the register specified by the second operand to the word data of the register specified by the third operand. And that result is stored in the register specified by the fourth operand.
- Syntax "adf*cnd* reg1, reg2, reg3"
  It compares the current flag condition with the flag condition indicated by the string in the *cnd*"part.
  If the values match, adds the word data of the register specified by the first operand to the word data of the register specified by the second operand. And 1 is added to the addition result and that result is stored in the register specified by the third operand.
  If the values not match, adds the word data of the register specified by the first operand to the word data of the register specified by the second operand. And that result is stored in the register specified by the third operand.

[Description]

- For the adf instruction, the assembler generates one adf machine instruction.
- For the adf*cnd* instruction, the assembler generates the corresponding adf instruction (see "Table 5.28 adfcnd Instruction List") and expands it to syntax "adf imm4, reg1, reg2, reg3".

Table 5.28 adf*cnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`adfgt`	((S xor OV) or Z) = 0	Greater than (signed)	`adf 0xF`
`adfge`	(S xor OV) = 0	Greater than or equal (signed)	`adf 0xE`
`adflt`	(S xor OV) = 1	Less than (signed)	`adf 0x6`
`adfle`	((S xor OV) or Z) = 1	Less than or equal (signed)	`adf 0x7`
`adfh`	(CY or Z) = 0	Higher (Greater than)	`adf 0xB`
`adfnl`	CY = 0	Not lower (Greater than or equal)	`adf 0x9`
`adfl`	CY = 1	Lower (Less than)	`adf 0x1`
`adfnh`	(CY or Z) = 1	Not higher (Less than or equal)	`adf 0x3`
`adfe`	Z = 1	Equal	`adf 0x2`
`adfne`	Z = 0	Not equal	`adf 0xA`

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`adfv`	OV = 1	Overflow	`adf 0x0`
`adfnv`	OV = 0	No overflow	`adf 0x8`
`adfn`	S = 1	Negative	`adf 0x4`
`adfp`	S = 0	Positive	`adf 0xC`
`adfc`	CY = 1	Carry	`adf 0x1`
`adfnc`	CY = 0	No carry	`adf 0x9`
`adfz`	Z = 1	Zero	`adf 0x2`
`adfnz`	Z = 0	Not zero	`adf 0xA`
`adft`	always 1	Always 1	`adf 0x5`

[Flag]

CY	1 if there is carry from MSB (Most Significant Bit), 0 if not
OV	1 if overflow occurred, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the adf instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

- If 0xD is specified as imm4 of the adf instruction, the following message is output, and assembly is stopped.

```
E0550261 : illegal condition code.
```

sub

Subtracts.

[Syntax]

- sub reg1, reg2
- sub imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "sub reg1, reg2"
  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.
- Syntax "sub imm, reg2"
  Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result into the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "sub reg1, reg2", the assembler generates one sub machine instruction.
- If the instruction is executed in syntax "sub imm, reg2", the assembler executes instruction expansion and generates one or more machine instructions[Note].

(a) 0

`sub 0, reg`	`sub r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`sub imm5, reg`	`mov imm5, r1` `sub r1, reg`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`sub imm16, reg`	`movea imm16, r0, r1` `sub 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

`sub imm, reg`	`movhi HIGHW(imm), r0, r1` `sub r1, reg`

Else

`sub imm, reg`	`mov imm, r1` `sub r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`sub $label, reg`	`movea $label, r0, r1` `sub 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

`sub #label, reg`	`mov #label, r1` `sub r1, reg`
`sub label, reg`	`mov label, r1` `sub r1, reg`
`sub $label, reg`	`mov $label, r1` `sub r1, reg`

Note The sub machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit),0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

subr

Subtracts reverse.

[Syntax]

- subr reg1, reg2
- subr imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "subr reg1, reg2"
  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand.
- Syntax "subr imm, reg2"
  Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result into the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "subr reg1, reg2", the assembler generates one subr machine instruction.
- If the instruction is executed in syntax "subr imm, reg2", the assembler executes instruction expansion and generates one or more machine instructions[Note].

(a) 0

`subr 0, reg`	`subr r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`subr imm5, reg`	`mov imm5, r1` `subr r1, reg`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`subr imm16, reg`	`movea imm16, r0, r1` `subr 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

`subr imm, reg`	`movhi HIGHW(imm), r0, r1` `subr r1, reg`

Else

`subr imm, reg`	`mov imm, r1` `subr r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`subr $label, reg`	`movea $label, r0, r1` `subr 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

`subr #label, reg`	`mov #label, r1` `subr r1, reg`
`subr label, reg`	`mov label, r1` `subr r1, reg`
`subr $label, reg`	`mov $label, r1` `subr r1, reg`

Note The subr machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if a borrow occurs from MSB (Most Significant Bit),0 if not

sbf

Subtracts on condition flag.

[Syntax]

- sbf imm4, reg1, reg2, reg3
- sbf*cnd* reg1, reg2, reg3

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits (0xD cannot be specified)

[Function]

- Syntax "sbf imm4, reg1, reg2, reg3"
  It compares the current flag condition with the flag condition indicated by the value of the lower 4 bits of the absolute expression (see "Table 5.29 sbfcnd Instruction List") specified by the first operand.
  If the values match, subtracts the word data of the register specified by the second operand from the word data of the register specified by the third operand. And 1 is subtracted from the subtraction result and that result is stored in the register specified by the fourth operand.
  If the values not match, subtracts the word data of the register specified by the second operand from the word data of the register specified by the third operand. And that result is stored in the register specified by the fourth operand.
- Syntax "sbf*cnd* reg1, reg2, reg3"
  It compares the current flag condition with the flag condition indicated by the string in the "*cnd*" part.
  If the values match, subtracts the word data of the register specified by the first operand from the word data of the register specified by the second operand. And 1 is subtracted from the subtraction result and that result is stored in the register specified by the third operand.
  If the values not match, subtracts the word data of the register specified by the first operand from the word data of the register specified by the second operand. And that result is stored in the register specified by the third operand.

[Description]

- For the sbf instruction, the assembler generates one sbf machine instruction.
- For the adcond instruction, the assembler generates the corresponding sbf instruction (see "Table 5.29 sbfcnd Instruction List") and expands it to syntax "sbf imm4, reg1, reg2, reg3".

Table 5.29 sbf*cnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`sbfgt`	((S xor OV) or Z) = 0	Greater than (signed)	`sbf 0xF`
`sbfge`	(S xor OV) = 0	Greater than or equal (signed)	`sbf 0xE`
`sbflt`	(S xor OV) = 1	Less than (signed)	`sbf 0x6`
`sbfle`	((S xor OV) or Z) = 1	Less than or equal (signed)	`sbf 0x7`
`sbfh`	(CY or Z) = 0	Higher (Greater than)	`sbf 0xB`
`sbfnl`	CY = 0	Not lower (Greater than or equal)	`sbf 0x9`
`sbfl`	CY = 1	Lower (Less than)	`sbf 0x1`
`sbfnh`	(CY or Z) = 1	Not higher (Less than or equal)	`sbf 0x3`
`sbfe`	Z = 1	Equal	`sbf 0x2`
`sbfne`	Z = 0	Not equal	`sbf 0xA`

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
sbfv	OV = 1	Overflow	sbf 0x0
sbfnv	OV = 0	No overflow	sbf 0x8
sbfn	S = 1	Negative	sbf 0x4
sbfp	S = 0	Positive	sbf 0xC
sbfc	CY = 1	Carry	sbf 0x1
sbfnc	CY = 0	No carry	sbf 0x9
sbfz	Z = 1	Zero	sbf 0x2
sbfnz	Z = 0	Not zero	sbf 0xA
sbft	always 1	Always 1	sbf 0x5

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit), 0 if not
OV	1 if overflow occurred, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the sbf instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

- If 0xD is specified as imm4 of the sbf instruction, the following message is output, and assembly is stopped.

```
E0550261 : illegal condition code.
```

mulh

Multiplies half-word.

[Syntax]

- mulh reg1, reg2
- mulh imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression

Note The assembler does not check whether the value of the expression exceeds 16 bits. The generated mulh instruction performs the operation by using the lower 16 bits.

[Function]

- Syntax "mulh reg1, reg2"
  Multiplies the value of the lower halfword data of the register specified by the first operand by the value of the lower halfword data of the register specified by the second operand as a signed value, and stores the result in the register specified by the second operand.
- Syntax "mulh imm, reg2"
  Multiplies the value of the lower halfword data of the absolute expression or relative expression specified by the first operand by the value of the lower halfword data of the register specified by the second operand as a signed value, and stores the result in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "mulh reg1, reg2", the assembler generates one mulh machine instruction.
- If the following is specified as imm in syntax "mulh imm, reg2", the assembler generates one mulh machine instruction[Note].

  (a) Absolute expression having a value in the range of -16 to +15

`mulh imm5, reg`	`mulh imm5, reg`

Note The mulh machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the first operand.

- If the following is specified for imm in syntax "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

`mulh imm16, reg`	`mulhi 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

`mulh imm, reg`	`movhi HIGHW(imm), r0, r1` `mulh r1, reg`

  Else

`mulh imm, reg`	`mov imm, r1` `mulh 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

`mulh !label, reg`	`mulhi !label, reg, reg`
`mulh %label, reg`	`mulhi %label, reg, reg`
`mulh $label, reg`	`mulhi $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

`mulh #label, reg`	`mov #label, r1` `mulh r1, reg`
`mulh label, reg`	`mov label, r1` `mulh r1, reg`
`mulh $label, reg`	`mov $label, r1` `mulh r1, reg`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the second operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

mulhi

Multiplies half-word Immediate.

[Syntax]

- mulhi imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

Note The assembler does not check whether the value of the expression exceeds 16 bits. The generated mulhi machine instruction performs the operation by using the lower 16 bits.

[Function]

Multiplies the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied specified by the first operand by the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the assembler generates one mulhi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`mulhi imm16, reg1, reg2`	`mulhi imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mulhi $label, reg1, reg2`	`mulhi $label, reg1, reg2`

(c) Relative expression having !label or %label

`mulhi !label, reg1, reg2`	`mulhi !label, reg1, reg2`
`mulhi %label, reg1, reg2`	`mulhi %label, reg1, reg2`

(d) Expression with HIGHW, LOWW, or HIGHW1

`mulhi imm16, reg1, reg2`	`mulhi imm16, reg1, reg2`

Note The mulhi machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the first operand.

- If the following is specified for imm, the assembler executes instruction expansion to generate two 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

`mulhi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `mulh reg1, reg2`

Else

`mulhi imm, reg1, reg2`	`mov imm, reg2` `mulh 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

`mulhi #label, reg1, reg2`	`mov #label, reg2` `mulh reg1, reg2`
`mulhi label, reg1, reg2`	`mov label, reg2` `mulh reg1, reg2`
`mulhi $label, reg1, reg2`	`mov $label, reg2` `mulh reg1, reg2`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the third operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

mul

Multiplies word.

[Syntax]

- mul reg1, reg2, reg3
- mul imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "mul reg1, reg2, reg3"
  Multiplies the register value specified by the first operand by the register value specified by the second operand as a signed value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.
- Syntax "mul imm, reg2, reg3"
  Multiplies the value of the absolute or relative expression specified by the first operand by the register value specified by the second operand as a signed value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

[Description]

- If the instruction is executed in syntax "mul reg1, reg2, reg3", the assembler generates one mul machine instruction.
- If the instruction is executed in syntax "mul imm, reg2, reg3", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

`mul 0, reg2, reg3`	`mul r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -256 to +255

`mul imm9, reg2, reg3`	`mul imm9, reg2, reg3`

(c) Absolute expression exceeding the range of -256 to +255, but within the range of -32,768 to +32,767

`mul imm16, reg2, reg3`	`movea imm16, r0, r1` `mul 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

`mul imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mul r1, reg2, reg3`

Else

`mul imm, reg2, reg3`	`mov imm, r1` `mul r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mul  $label, reg2, reg3`	`movea  $label, r0, r1` `mul  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

`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`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

mulu

Multiplies unsigned word.

[Syntax]

- mulu reg1, reg2, reg3
- mulu imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "mulu reg1, reg2, reg3"
  Multiplies the register value specified by the first operand by the register value specified by the second operand as an unsigned value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.
- Syntax "mulu imm, reg2, reg3"
  Multiplies the value of the absolute or relative expression specified by the first operand by the register value specified by the second operand as an unsigned value and stores the lower 32 bits of the result in the register specified by the second operand, and the higher 32 bits in the register specified by the third operand. If the same register is specified by the second and third operands, the higher 32 bits of the multiplication result are stored in that register.

[Description]

- If the instruction is executed in syntax "mulu reg1, reg2, reg3", the assembler generates one mulu machine instruction.
- If the instruction is executed in syntax "mulu imm, reg2, reg3", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

`mulu 0, reg2, reg3`	`mulu r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of 0 to 511

`mulu imm9, reg2, reg3`	`mulu imm9, reg2, reg3`

(c) Absolute expression exceeding the range of -256 to +255, but within the range of -32,768 to +32,767

`mulu imm16, reg2, reg3`	`movea imm16, r0, r1` `mulu 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

`mulu imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mulu r1, reg2, reg3`

Else

`mulu imm, reg2, reg3`	`mov imm, r1` `mulu r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`mulu  $label, reg2, reg3`	`movea  $label, r0, r1` `mulu  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

`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`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

mac

Multiplies and adds signed word data.

[Syntax]

- mac reg1, reg2, reg3, reg4

[Function]

Adds the multiplication result of the general-purpose register reg2 word data and the general-purpose register reg1 word data with the 64-bit data made up of general-purpose register reg3 as the lower 32 bits and general-purpose register reg3+1 (for example, if reg3 were r6, "reg3+1" would be r7) as the upper 32 bits, and stores the upper 32 bits of that result (64-bit data) in general-purpose register reg4+1 and the lower 32 bits in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are treated as 32-bit signed integers.

General-purpose registers reg1, reg2, reg3, and reg3+1 are unaffected.

[Description]

The assembler generates one mac machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- The general-purpose registers that can be specified to reg3 or reg4 are limited to even numbered registers (r0, r2, r4, ..., r30). When specifying an odd numbered register, the following message is output, and assembly continues, specifying the register as an even numbered register (r0, r2, r4, ..., r30).

```
W0550026 : illegal register number, aligned odd register(rXX) to be even register(rYY).
```

macu

Multiply and adds unsigned word data.

[Syntax]

- macu reg1, reg2, reg3, reg4

[Function]

Adds the multiplication result of the general-purpose register reg2 word data and the general-purpose register reg1 word data with the 64-bit data made up of general-purpose register reg3 as the lower 32 bits and general-purpose register reg3+1 (for example, if reg3 were r6, "reg3+1" would be r7) as the upper 32 bits, and stores the upper 32 bits of that result (64-bit data) in general-purpose register reg4+1 and the lower 32 bits in general-purpose register reg4.

The contents of general-purpose registers reg1 and reg2 are treated as 32-bit unsigned integers.

General-purpose registers reg1, reg2, reg3, and reg3+1 are unaffected.

[Description]

The assembler generates one macu machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- The general-purpose registers that can be specified to reg3 or reg4 are limited to even numbered registers (r0, r2, r4, ..., r30). When specifying an odd numbered register, the following message is output, and assembly continues, specifying the register as an even numbered register (r0, r2, r4, ..., r30).

```
W0550026 : illegal register number, aligned odd register(rXX) to be even register(rYY).
```

divh

Divides half-word.

[Syntax]

- divh reg1, reg2
- divh imm, reg2
- divh reg1, reg2, reg3
- divh imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression

Note The assembler does not check whether the value of the expression exceeds 16 bits. The generated machine instruction performs execution using the lower 16 bits.

[Function]

- Syntax "divh reg1, reg2"
  Divides the register value specified by the second operand by the value of the lower halfword data of the register specified by the first operand as a signed value, and stores the quotient in the register specified by the second operand.
- Syntax "divh imm, reg2"
  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand.
- Syntax "divh reg1, reg2, reg3"
  Divides the register value specified by the second operand by the value of the lower halfword data of the register specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.
- Syntax "divh imm, reg2, reg3"
  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntaxes "divh reg1, reg2" and "divh reg1, reg2, reg3", the assembler generates one divh machine instruction.
- If the instruction is executed in syntax "divh imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) Absolute expression having a value of other than 0 within the range of -16 to +15

`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

`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

`divh imm, reg`	`movhi HIGHW(imm), r0, r1` `divh r1, reg`

Else

`divh imm, reg`	`mov imm, r1` `divh r1, reg`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

Note The divh machine instruction does not take an immediate value as an operand.

- If the instruction is executed in syntax "divh imm, reg2, reg3", the assembler executes instruction expansion to generate one or more machine instructions.

(a) 0

`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

`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

`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

`divh imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `divh r1, reg2, reg3`

Else

`divh imm, reg2, reg3`	`mov imm, r1` `divh r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

[Flag]

CY	---
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If r0 is specified by the first operand in syntax "divh reg1, reg2", the assembler outputs the following message and stops assembling.

E0550239 : Illegal operand (cannot use r0 as source in RH850 mode).
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

- If r0 is specified by the second operand (reg2) in syntaxes "divh imm, reg2", the assembler outputs the message and stops assembling.

E0550239 : Illegal operand (cannot use r0 as source in RH850 mode).
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

- If 0 is specified by the second operand (imm) in syntaxes "divh imm, reg2", the assembler outputs the message and stops assembling.

E0550239 : Illegal operand (cannot use r0 as source in RH850 mode).

div

Divides word.

[Syntax]

- div reg1, reg2, reg3
- div imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "div reg1, reg2, reg3"
  Divides the register value specified by the second operand by the register value specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.
- Syntax "div imm, reg2, reg3"
  Divides the register value specified by the second operand by the value of the absolute or relative expression specified by the first operand as a signed value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax "div reg1, reg2, reg3", the assembler generates one div machine instruction.
- If the instruction is executed in syntax "div imm, reg2, reg3", the assembler executes instruction expansion to generate two or more machine instructions[Note].

(a) 0

`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

`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

`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

`div imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `div r1, reg2, reg3`

Else

`div imm, reg2, reg3`	`mov imm, r1` `div r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

Note The div machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

divhu

Divides unsigned half-word.

[Syntax]

- divhu reg1, reg2, reg3
- divhu imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 16 bits[Note]
- Relative expression

Note The assembler does not check whether the value of the expression exceeds 16 bits. The generated machine instruction uses only the lower 16 bits for execution.

[Function]

- Syntax "divhu reg1, reg2, reg3"
  Divides the register value specified by the second operand by the value of the lower halfword data of the register value specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.
- Syntax "divhu imm, reg2, reg3"
  Divides the register value specified by the second operand by the value of the lower halfword data of the absolute or relative expression specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax "divhu reg1, reg2, reg3", the assembler generates one divhu machine instruction.
- If the instruction is executed in syntax "divhu imm, reg2, reg3", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`divhu  0, reg2, reg3`	`divhu  r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`divhu  imm5, reg2, reg3`	`mov  imm5, r1` `divhu  r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divhu  imm16, reg2, reg3`	`movea  imm16, r0, r1` `divhu  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

`divhu  imm, reg2, reg3`	`movhi  HIGHW(imm), r0, r1` `divhu  r1, reg2, reg3`

Else

`divhu imm, reg2, reg3`	`mov imm, r1` `divhu r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`divhu $label, reg2, reg3`	`movea $label, r0, r1` `divhu 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

`divhu #label, reg2, reg3`	`mov #label, r1` `divhu r1, reg2, reg3`
`divhu label, reg2, reg3`	`mov label, r1` `divhu r1, reg2, reg3`
`divhu $label, reg2, reg3`	`mov $label, r1` `divhu r1, reg2, reg3`

Note The divhu machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

divu

Divides unsigned word.

[Syntax]

- divu reg1, reg2, reg3
- divu imm, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "divu reg1, reg2, reg3"
  Divides the register value specified by the second operand by the register value specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.
- Syntax "divu imm, reg2, reg3"
  Divides the register value specified by the second operand by the value of the absolute or relative expression specified by the first operand as an unsigned value and stores the quotient in the register specified by the second operand, and the remainder in the register specified by the third operand. If the same register is specified by the second and third operands, the remainder is stored in that register.

[Description]

- If the instruction is executed in syntax "divu reg1, reg2, reg3", the assembler generates one divu machine instruction.
- If the instruction is executed in syntax "divu imm, reg2, reg3", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`divu 0, reg2, reg3`	`divu r0, reg2, reg3`

(b) Absolute expression having a value of other than 0 whithin the range of -16 to +15

`divu imm5, reg2, reg3`	`mov imm5, r1` `divu r1, reg2, reg3`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`divu imm16, reg2, reg3`	`movea imm16, r0, r1` `divu 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

`divu imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `divu r1, reg2, reg3`

Else

`divu imm, reg2, reg3`	`mov imm, r1` `divu r1, reg2, reg3`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`divu  $label, reg2, reg3`	`movea  $label, r0, r1` `divu  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

`divu  #label, reg2, reg3`	`mov  #label, r1` `divu  r1, reg2, reg3`
`divu  label, reg2, reg3`	`mov  label, r1` `divu  r1, reg2, reg3`
`divu  $label, reg2, reg3`	`mov  $label, r1` `divu  r1, reg2, reg3`

Note The divu machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

divq

Division of (signed) word data (variable steps) (Divide Word Quickly)

[Syntax]

- divq reg1, reg2, reg3

[Function]

Divides the word data in general-purpose register reg2 by the word data in general-purpose register reg1, stores the quotient in reg2, and stores the remainder in general-purpose register reg3. General-purpose register reg1 is not affected.
The minimum number of steps required for division is determined from the values in reg1 and reg2, then this operation is executed.
When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Description]

- The assembler generates one divq machine instruction.

[Flag]

CY	---
OV	"1" when overflow occurs; otherwise, "0"
S	"1" when operation result quotient is a negative value; otherwise, "0"
Z	"1" when operation result quotient is a "0"; otherwise, "0"
SAT	---

divqu

Division of (unsigned) word data (variable steps) (Divide Word Unsigned Quickly)

[Syntax]

- divqu reg1, reg2, reg3

[Function]

Divides the word data in general-purpose register reg2 by the word data in general-purpose register reg1, stores the quotient in reg2, and stores the remainder in general-purpose register reg3. General-purpose register reg1 is not affected.
The minimum number of steps required for division is determined from the values in reg1 and reg2, then this operation is executed.
When division by zero occurs, an overflow results and all operation results except for the OV flag are undefined.

[Description]

- The assembler generates one divqu machine instruction.

[Flag]

CY	---
OV	"1" when overflow occurs; otherwise, "0"
S	"1" when the MSB in the word data of the operation result quotient is a negative value; otherwise, "0"
Z	"1" when operation result quotient is a "0"; otherwise, "0"
SAT	---

cmp

Compares.

[Syntax]

- cmp reg1, reg2
- cmp imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "cmp reg1, reg2"
  Compares the value of the register specified by the first operand with the value of the register specified by the second operand, and indicates the result using a flag. Comparison is performed by subtracting the value of the register specified by the first operand from the value of the register specified by the second operand.
- Syntax "cmp imm, reg2"
  Compares the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and indicates the result using a flag. Comparison is performed by subtracting the value of the register specified by the first operand from the value of the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "cmp reg1, reg2", the assembler generates one cmp machine instruction.
- If the following is specified as imm in syntax "cmp imm, reg2", the assembler generates one cmp machine instruction-Note.

  (a) Absolute expression having a value in the range of -16 to +15

`cmp imm5, reg`	`cmp imm5, reg`

Note The cmp machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the first operand.

- If the following is specified as imm in syntax "cmp 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

`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

`cmp imm, reg`	`movhi HIGHW(imm), r0, r1` `cmp r1, reg`

  Else

`cmp imm, reg`	`mov imm, r1` `cmp r1, reg`

(c) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit),0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

mov

Moves.

[Syntax]

- mov reg1, reg2
- mov imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "mov reg1, reg2"
  Stores the value of the register specified by the first operand in the register specified by the second operand.
- Syntax "mov imm, reg2"
  Stores the value of the absolute expression or relative expression specified by the first operand in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "mov reg1, reg2", the assembler generates one mov machine instruction.
- If the following is specified as imm in syntax "mov imm, reg2", the assembler generates one mov machine instruction-Note.

  (a) Absolute expression having a value in the range of -16 to +15

`mov imm5, reg`	`mov imm5, reg`

Note The mov machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the first operand.

- If the following is specified as imm in syntax "mov 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

`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

`mov imm, reg`	`movhi HIGHW(imm), r0, reg`

  Else[Note]

`mov imm, reg`	`mov imm, 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

`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]

`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.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value in the range between -16 and 15 is specified by the first operand and r0 is specified by the second operand of syntax "mov imm, reg2", or r0 is specified by the second operand of syntax "mov reg1, reg2", the assembler outputs the following message and stops assembling.

```
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

- For the "mov imm, reg2" instruction, if an absolute expression exceeding the range of -32768 to 32767, a relative expression with a #label or label, and a relative expression with label $label without a definition in an sdata/sbss attribute section is specified in the first operand, then the assembler performs instruction expansion, and converts the 16-bit mov instruction into a 48-bit mov instruction. If you prevent instruction expansion using the .option nomacro directive and this instruction expansion occurs, then the following message is output, and the assembler halts.

```
E0550249 : illegal syntax
```

movea

Moves execution address.

[Syntax]

- movea imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

Adds the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied, specified by the first operand, to the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the assembler generates one movea machine instruction[Note].
- If r0 is specified by reg1, the assembler recognizes specified syntax "mov imm, reg2".

(a) Absolute expression having a value in the range of -32,768 to +32,767

`movea imm16, reg1, reg2`	`movea imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`movea $label, reg1, reg2`	`movea $label, reg1, reg2`

(c) Relative expression having !label or %label

`movea !label, reg1, reg2`	`movea !label, reg1, reg2`
`movea %label, reg1, reg2`	`movea %label, reg1, reg2`

(d) Expression with HIGHW, LOWW, or HIGHW1

`movea imm16, reg1, reg2`	`movea imm16, reg1, reg2`

Note The movea machine instruction takes an immediate value in a range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the first operand.

- 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

`movea imm, reg1, reg2`	`movhi HIGHW(imm), reg1, reg2`

Else

`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

`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`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the third operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

movhi

Moves higher half-word.

[Syntax]

- movhi imm16, reg1, reg2

The following can be specified for imm16:

- Absolute expression having a value of up to 16 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

Adds word data for which the higher 16 bits are specified by the first operand and the lower 16 bits are 0, to the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

The assembler generates one movhi machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 65,535 is specified as imm16, the assembler outputs the following message and stops assembling.

```
E0550231 : illegal operand (range error in immediate)
```

- If r0 is specified by the third operand, the assembler outputs the following message and stops assembling.

```
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

mov32

Moves 32-bit data.

[Syntax]

- mov32 imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression[Note]

Note If the operand is a relative expression, HIGHW, LOWW, and HIGHW1 cannot be used.

[Function]

Stores the value of the absolute or relative expression specified as the first operand in the register specified as the second operand.

[Description]

The assembler generates one 48-bit machine language mov instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

cmov

Moves data depending on the flag condition.

[Syntax]

- cmov imm4, reg1, reg2, reg3
- cmov imm4, imm, reg2, reg3
- cmov*cnd* reg1, reg2, reg3
- cmov*cnd* imm, reg2, reg3

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits[Note]

Note The cmov machine instruction takes an immediate value in the range of 0 to 15 (0x0 to 0xF) as the first operand.

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "cmov imm4, reg1, reg2, reg3"
  Compares the flag condition indicated by the value of the lower 4 bits of the value of the constant expression specified by the first operand with the current flag condition. If a match is found, the register value specified by the second operand is stored in the register specified by the fourth operand; otherwise, the register value specified by the third operand is stored in the register specified by the fourth operand.
- Syntax "cmov imm4, imm, reg2, reg3"
  Compares the flag condition indicated by the value of the lower 4 bits of the constant expression specified by the first operand with the current flag condition. If a match is found, the value of the absolute expression specified by the second operand is stored in the register specified by the fourth operand; otherwise, the register value specified by the third operand is stored in the register specified by the fourth operand.
- Syntax "cmov*cnd* reg1, ret2, reg3"
  Compares the flag condition indicated by string *cnd* with the current flag condition. If a match is found, the register value specified by the first operand is stored in the register specified by the third operand; otherwise, the register value specified by the second operand is stored in the register specified by the third operand.
- Syntax "cmov*cnd* imm, reg2, reg3"
  Compares the flag condition indicated by string *cnd* with the current flag condition. If a match is found, the value of the absolute expression specified by the first operand is stored in the register specified by the third operand; otherwise, the register value specified by the second operand is stored in the register specified by the third operand.

Table 5.30 cmov*cnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`cmovgt`	((S xor OV) or Z) = 0	Greater than (signed)	`cmov 0xF`
`cmovge`	(S xor OV) = 0	Greater than or equal (signed)	`cmov 0xE`
`cmovlt`	(S xor OV) = 1	Less than (signed)	`cmov 0x6`
`cmovle`	((S xor OV) or Z) = 1	Less than or equal (signed)	`cmov 0x7`
`cmovh`	(CY or Z) = 0	Higher (Greater than)	`cmov 0xB`
`cmovnl`	CY = 0	Not lower (Greater than or equal)	`cmov 0x9`

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
cmovl	CY = 1	Lower (Less than)	cmov 0x1
cmovnh	(CY or Z) = 1	Not higher (Less than or equal)	cmov 0x3
cmove	Z = 1	Equal	cmov 0x2
cmovne	Z = 0	Not equal	cmov 0xA
cmovv	OV = 1	Overflow	cmov 0x0
cmovnv	OV = 0	No overflow	cmov 0x8
cmovn	S = 1	Negative	cmov 0x4
cmovp	S = 0	Positive	cmov 0xC
cmovc	CY = 1	Carry	cmov 0x1
cmovnc	CY = 0	No carry	cmov 0x9
cmovz	Z = 1	Zero	cmov 0x2
cmovnz	Z = 0	Not zero	cmov 0xA
cmovt	always 1	Always 1	cmov 0x5
cmovsa	SAT = 1	Saturated	cmov 0xD

[Description]

- If the instruction is executed in syntax "cmov imm4, reg1, reg2, reg3", the assembler generates one cmov machine instruction[Note].

Note The cmov machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the second operand.

- If the following is specified as imm in syntax "cmov imm4, imm, reg2, reg3", the assembler generates one cmov machine instruction.

(a) Absolute expression having a value in the range of -16 to +15

```
cmov    imm4, imm5, reg2, reg3
```

```
cmov    imm4, imm5, reg2, reg3
```

- If the following is specified as imm in syntax "cmov imm4, imm, reg2, reg3", the assembler executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

```
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

```
cmov    imm4, imm, reg2, reg3
```

```
movhi   HIGHW(imm), r0, r1
cmov    imm4, r1, reg2, reg3
```

Else

```
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

`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

`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`

- If the instruction is executed in syntax "cmov*cnd* reg1, ret2, reg3", the assembler generates the corresponding cmov instruction (see "Table 5.30 cmovcnd Instruction List") and expands it to syntax "cmov imm4, reg1, reg2, reg3".
- If the following is specified as imm in syntax "cmov*cnd* imm, reg2, reg3", the assembler generates the corresponding cmov instruction (see "Table 5.30 cmovcnd Instruction List") and expands it to syntax "cmov imm, reg2, reg3".

(a) Absolute expression having a value in the range of -16 to +15

- If the following is specified as imm in syntax "cmov*cnd* imm, reg2, reg3", the assembler executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`cmov`*cnd* `imm16, reg2, reg3`	`movea imm16, r0, r1` `cmov`*cnd* `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

`cmov`*cnd* `imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `cmov`*cnd* `r1, reg2, reg3`

Else

`cmov`*cnd* `imm, reg2, reg3`	`mov imm, r1` `cmov`*cnd* `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

`cmov`*cnd* `#label, reg2, reg3`	`mov #label, r1` `cmov`*cnd* `r1, reg2, reg3`
`cmov`*cnd* `label, reg2, reg3`	`mov label, r1` `cmov`*cnd* `r1, reg2, reg3`
`cmov`*cnd* `$label, reg2, reg3`	`mov $label, r1` `cmov`*cnd* `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

`cmov`*`cnd`* `!label, reg2, reg3`	`movea  !label, r0, r1` `cmov`*`cnd`* `r1, reg2, reg3`
`cmov`*`cnd`* `%label, reg2, reg3`	`movea  %label, r0, r1` `cmov`*`cnd`* `r1, reg2, reg3`
`cmov`*`cnd`* `$label, reg2, reg3`	`movea  $label, r0, r1` `cmov`*`cnd`* `r1, reg2, reg3`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If a constant expression having a value exceeding 4 bits is specified as imm4 of the cmov instruction, the assembler outputs the following message.
  If the value exceeds 4 bits, the assembler masks the value with 0xF and continues assembling.

W0550011 : illegal operand (range error in immediate)

setf

Sets flag condition.

[Syntax]

- setf imm4, reg
- setf*cnd* reg

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits

[Function]

- Syntax "setf imm4, reg"
  Compares the status of the flag specified by the value of the lower 4 bits of the absolute expression specified by the first operand with the current flag condition. If they are found to match, 1 is stored in the register specified by the second operand; otherwise, 0 is stored in the register specified by the second operand.
- Syntax "setf*cnd* reg"
  Compares the status of the flag indicated by string *cnd* with the current flag condition. If they are found to match, 1 is stored in the register specified by the second operand; otherwise, 0 is stored in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax"setf imm4, reg",the assembler generates one satf machine instruction.
- If the instruction is executed in syntax "setf*cnd* reg", the assembler generates the corresponding setf instruction (see "Table 5.31 setfcnd Instruction List") and expands it to syntax "setf imm4, reg".

Table 5.31 setf*cnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`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`

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`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`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the setf instruction, the assembler outputs the following message and continues assembling using four low-order bits of a specified value.

W0550011: illegal operand (range error in immediate).

sasf

Sets the flag condition after a logical left shift.

[Syntax]

- sasf imm4, reg
- sasf*cnd* reg

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits

[Function]

- Syntax "sasf imm4, reg"
  Compares the flag condition indicated by the value of the lower 4 bits of the absolute expression specified by the first operand with the current flag condition. If a match is found, the contents of the register specified by the second operand are shifted logically 1 bit to the left and ORed with 1, and the result stored in the register specified by the second operand; otherwise, the contents of the register specified by the second operand are logically shifted 1 bit to the left and the result stored in the register specified by the second operand.
- Syntax "sasf*cnd* reg"
  Compares the flag condition indicated by string *cnd* with the current flag condition. If a match is found, the contents of the register specified by the second operand are shifted logically 1 bit to the left and ORed with 1, and the result stored in the register specified by the second operand; otherwise, the contents of the register specified by the second operand are shifted logically 1 bit to the left and the result stored in the register specified by the second operand.

[Description]

- If the instruction is executed in syntax "sasf imm4, reg", the assembler generates one sasf machine instruction.
- If the instruction is executed in syntax "sasf*cnd* reg", the assembler generates the corresponding sasf instruction (see "Table 5.32 sasfcnd Instruction List") and expands it to syntax "sasf imm4, reg".

Table 5.32 sasf*cnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
`sasfgt`	((S xor OV) or Z) = 0	Greater than (signed)	`sasf 0xF`
`sasfge`	(S xor OV) = 0	Greater than or equal (signed)	`sasf 0xE`
`sasflt`	(S xor OV) = 1	Less than (signed)	`sasf 0x6`
`sasfle`	((S xor OV) or Z) = 1	Less than or equal (signed)	`sasf 0x7`
`sasfh`	(CY or Z) = 0	Higher (Greater than)	`sasf 0xB`
`sasfnl`	CY = 0	Not lower (Greater than or equal)	`sasf 0x9`
`sasfl`	CY = 1	Lower (Less than)	`sasf 0x1`
`sasfnh`	(CY or Z) = 1	Not higher (Less than or equal)	`sasf 0x3`
`sasfe`	Z = 1	Equal	`sasf 0x2`
`sasfne`	Z = 0	Not equal	`sasf 0xA`
`sasfv`	OV = 1	Overflow	`sasf 0x0`
`sasfnv`	OV = 0	No overflow	`sasf 0x8`
`sasfn`	S = 1	Negative	`sasf 0x4`

Instruction	Flag Condition	Meaning of Flag Condition	Instruction Expansion
sasfp	S = 0	Positive	sasf 0xC
sasfc	CY = 1	Carry	sasf 0x1
sasfnc	CY = 0	No carry	sasf 0x9
sasfz	Z = 1	Zero	sasf 0x2
sasfnz	Z = 0	Not zero	sasf 0xA
sasft	always 1	Always 1	sasf 0x5
sasfsa	SAT = 1	Saturated	sasf 0xD

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the sasf instruction, the assembler outputs the following message and continues assembling using four low-order bits of a specified value.

W0550011 : illegal operand (range error in immediate).

bins

Insert bit in register (Bitfield Insert)

[Syntax]

- bins reg1, pos, width, reg2

The following can be specified as ipos and width:

- Absolute expression having a value of up to 5 bits

[Function]

Loads the lower width bits in general-purpose register reg1 and stores them from the bit position bit pos + width - 1 in the specified field in general-purpose register reg2 in bit pos. This instruction does not affect any fields in general-purpose register reg2 except the specified field, nor does it affect general-purpose register reg1.

[Description]

- The assembler generates one bins machine instruction.

[Flag]

CY	---
OV	0
S	"1" if the word data MSB of the result is "1", otherwise, "0"
Z	"1" if the operation result is "0"; otherwise, "0"
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified as pos or width, the assembler outputs the following message.

```
W0550011 : Illegal operand (range error in immediate).
```

rotl

Rotate (Rotate Left)

[Syntax]

- rotl imm, reg2, reg3
- rotl reg1, reg2, reg3

The following can be specified as imm:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "rotl imm, reg2, reg3"
  Rotates the word data of general-purpose register reg2 to the left by the specified shift amount, which is indicated by a 5-bit immediate value zero-extended to word length. The result is written to general-purpose register reg3. General-purpose register reg2 is not affected.
- Syntax "rotl reg1, reg2, reg3"
  Rotates the word data of general-purpose register reg2 to the left by the specified shift amount indicated by the lower 5 bits of general-purpose register reg1. The result is written to general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.

[Description]

- If the following is specified as imm in syntax "rotl imm, reg2, reg3", the assembler generates one rotl[Note] machine instruction.

  (a) Absolute expression having a value in the range of 0 to 31

Note The rotl machine instruction takes a register or immediate value in the range of 0 to 31 (0 to 0x1F) as the first operand.

- Syntax "rotl reg1, reg2, reg3"
  The assembler generates one rotl machine instruction.

[Flag]

CY	"1" if operation result bit 0 is "1"; otherwise "0", including if the rotate amount is "0"
OV	0
S	"1" if the operation result is negative; otherwise, "0"
Z	"1" if the operation result is "0"; otherwise, "0"
SAT	---

5.9.3 Saturated operation instructions

This section describes the saturated operation instructions. Next table lists the instructions described in this section. See the RH850 product user's manual and architecture edition for details.

Table 5.33 Saturated Operation Instructions

Instruction	Meaning
satadd	Adds saturated
satsub	Subtracts saturated
satsubi	Subtracts saturated (immediate)
satsubr	Subtracts reverse saturated

satadd

Adda saturated.

[Syntax]

- satadd reg1, reg2
- satadd imm, reg2
- satadd reg1, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "satadd reg1, reg2"
  Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.
- Syntax "satadd imm, reg2"
  Adds the value of the absolute expression or relative expression specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.
- Syntax "satadd reg1, reg2, reg3"
  Adds the value of the register specified by the first operand to the value of the register specified by the second operand, and stores the result in the register specified by the third operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax "satadd reg1, reg2" or "satadd reg1, reg2, reg3", the assembler generates one satadd machine instruction.
- If the following is specified for imm in syntax "satadd imm, reg2", the assembler generates one satadd machine instruction[Note].

  (a) Absolute expression having a value in the range of -16 to +15

`satadd imm5, reg`	`satadd imm5, reg`

Note The satadd machine instruction takes a register or immediate value in the range of -16 to +15 (0xFFFFFFF0 to 0xF) as the first operand.

- If the following is specified for imm in syntax "satadd 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

`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

`satadd imm, reg`	`movhi HIGHW(imm), r0, r1` `satadd r1, reg`

Else

`satadd imm, reg`	`mov imm, r1` `satadd r1, reg`

(c) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If the instruction is executed in syntax "satadd reg1, reg2" or "satadd imm, reg2", if r0 is specified as the second operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

satsub

Subtracts saturated.

[Syntax]

- satsub reg1, reg2
- satsub imm, reg2
- satsub reg1, reg2, reg3

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "satsub reg1, reg2"
  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the third operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1
- Syntax "satsub imm, reg2"
  Subtracts the value of the absolute expression or relative expression specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.
- Syntax "satsub reg1, reg2, reg3"
  Subtracts the value of the register specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax "satsub reg1, reg2" or "satsub reg1, reg2, reg3", the assembler generates one satsub machine instruction.
- If the instruction is executed in syntax "satsub imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`satsub 0, reg`	`satsub r0, reg`

(b) Absolute expression having a value in the range of -32,768 to +32,767

`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

`satsub imm, reg`	`movhi HIGHW(imm), r0, r1` `satsub r1, reg`

Else

`satsub imm, reg`	`mov imm, r1` `satsub r1, reg`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

Note The satsub machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If the instruction is executed in syntax "satsub reg1, reg2" or "satsub imm, reg2", if r0 is specified as the second operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

satsubi

Subtracts saturated (immediate).

[Syntax]

- satsubi imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

Subtracts the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied specified by the first operand from the value of the register specified by the second operand, and stores the result in the register specified by the third operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the third operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the third operand. In both cases, the SAT flag is set to 1.

[Description]

- If the following is specified for imm, the assembler generates one satsubi machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satsubi $label, reg1, reg2`	`satsubi $label, reg1, reg2`

(c) Relative expression having !label or %label

`satsubi !label, reg1, reg2`	`satsubi !label, reg1, reg2`
`satsubi %label, reg1, reg2`	`satsubi %label, reg1, reg2`

(d) Expression with HIGHW, LOWW, or HIGHW1

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

Note The satsubi machine instruction takes an immediate value, in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF), as the first operand.

- 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

<1> If all the lower 16 bits of the value of imm are 0

`satsubi imm, reg1, reg2`	`movhi  HIGHW(imm), r0, reg2` `satsubr reg1, reg2`

<2> Else

`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

`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`

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If r0 is specified by the second operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

satsubr

Subtracts reverse saturated.

[Syntax]

- satsubr reg1, reg2
- satsubr imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "satsubr reg1, reg2"
  Subtracts the value of the register specified by the second operand from the value of the register specified by the first operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.
- Syntax "satsubr imm, reg2"
  Subtracts the value of the register specified by the second operand from the value of the absolute expression or relative expression specified by the first operand, and stores the result in the register specified by the second operand. If the result exceeds the maximum positive value of 0x7FFFFFFF, however, 0x7FFFFFFF is stored in the register specified by the second operand. Likewise, if the result exceeds the maximum negative value of 0x80000000, 0x80000000 is stored in the register specified by the second operand. In both cases, the SAT flag is set to 1.

[Description]

- If the instruction is executed in syntax "satsubr reg1, reg2", the assembler generates one satsubr machine instruction.
- If the instruction is executed in syntax "satsubr imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`satsubr 0, reg`	`satsubr r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`satsubr imm5, reg`	`mov  imm5, r1` `satsubr r1, reg`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`satsubr imm16, reg`	`movea  imm16, r0, r1` `satsubr 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

`satsubr imm, reg`	`movhi  HIGHW(imm), r0, r1` `satsubr r1, reg`

Else

`satsubr imm, reg`	`mov  imm, r1` `satsubr r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`satsubr $label, reg`	`movea  $label, r0, r1` `satsubr 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

`satsubr #label, reg`	`mov  #label, r1` `satsubr r1, reg`
`satsubr label, reg`	`mov  label, r1` `satsubr r1, reg`
`satsubr $label, reg`	`mov  $label, r1` `satsubr r1, reg`

Note The satsubr machine instruction does not take an immediate value as an operand.

[Flag]

CY	1 if a borrow occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	1 if OV = 1, - if not

[Caution]

- If r0 is specified by the second operand, the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

5.9.4 Logical instructions

This section describes the logical instructions. Next table lists the instructions described in this section.
See the RH850 product user's manual and architecture edition for details.

Table 5.34 Logical Instructions

Instruction	Meanings
or	Logical sum
ori	Logical sum (immediate)
xor	Exclusive OR
xori	Exclusive OR (immediate)
and	Logical product
andi	Logical product (immediate)
not	Logical negation (takes 1's complement)
shr	Logical right shift
sar	Arithmetic right shift
shl	Logical left shift
sxb	Sign extension of byte data
sxh	Sign extension of 2-byte data
zxb	Zero extension of byte data
zxh	Zero extension of 2-byte data
bsh	Byte swap of half-word data
bsw	Byte swap of word data
hsh	Half-word swap of half-word data
hsw	Half-word swap of word data
tst	Test
sch0l	Bit (0) search from MSB side
sch0r	Bit (0) search from LSB side
sch1l	Bit (1) search from MSB side
sch1r	Bit (1) search from LSB side

or

Logical sum.

[Syntax]

- or reg1, reg2
- or imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "or reg1, reg2"
  ORs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.
- Syntax "or imm, reg2"
  ORs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax "or reg1, reg2", the assembler generates one or machine instruction.
- When this instruction is executed in syntax "or imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`or 0, reg`	`or r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

`or imm16, reg`	`ori imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`or imm5, reg`	`mov imm5, r1` `or r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

`or imm16, reg`	`movea imm16, r0, r1` `or r1, reg`

(e) Absolute expression exceeding the above ranges
If all the lower 16 bits of the value of imm are 0

`or imm, reg`	`movhi HIGHW(imm), r0, r1` `or r1, reg`

Else

`or imm, reg`	`mov imm, r1` `or r1, reg`

(f) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`or  $label, reg`	`movea  $label, r0, r1` `or  r1, reg`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`or  #label, reg`	`mov  #label, r1` `or  r1, reg`
`or  label, reg`	`mov  label, r1` `or  r1, reg`
`or  $label, reg`	`mov  $label, r1` `or  r1, reg`

Note The or machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

ori

Logical sum (immediate).

[Syntax]

- ori imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

ORs the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the assembler generates one ori machine instruction[Note 1].

(a) Absolute expression having a value in the range of 0 to 65,535

`ori imm16, reg1, reg2`	`ori imm16, reg1, reg2`

(b) Relative expression having !label or %label[Note 2]

`ori !label, reg1, reg2`	`ori !label, reg1, reg2`
`ori %label, reg1, reg2`	`ori %label, reg1, reg2`

(c) Expression with HIGHW, LOWW, or HIGHW1

`ori imm16, reg1, reg2`	`ori imm16, reg1, reg2`

Note 1. The ori machine instruction takes an immediate value of 0 to 65,535 (0 to 0xFFFF) as the first operand.

Note 2. Values from -32768 to 32767 are specifiable for !label and $label and handled as immediate values (padded with zeros).

- 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 in the range of -16 to -1
If reg2 is r0

`ori imm5, reg1, r0`	`mov imm5, r1` `or reg1, r1`

Else

`ori imm5, reg1, reg2`	`mov imm5, reg2` `or reg1, reg2`

(b) Absolute expression having a value in the range of -32,768 to -17
If reg2 is r0

`ori imm16, reg1, r0`	`movea imm16, r0, r1` `or reg1, r1`

Else

`ori imm16, reg1, reg2`	`movea imm16, r0, reg2` `or 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

`ori imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `or reg1, r1`

Else

`ori imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `or reg1, reg2`

<2> Else
If reg2 is r0

`ori imm, reg1, r0`	`mov imm, r1` `or reg1, r1`

Else

`ori imm, reg1, reg2`	`mov imm, reg2` `or reg1, reg2`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section
If reg2 is r0

`ori $label, reg1, r0`	`movea $label, r0, r1` `or reg1, r1`

Else

`ori $label, reg1, reg2`	`movea $label, r0, reg2` `or 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

`ori #label, reg1, r0`	`mov #label, r1` `or reg1, r1`
`ori label, reg1, r0`	`mov label, r1` `or reg1, r1`
`ori $label, reg1, r0`	`mov $label, r1` `or reg1, r1`

Else

`ori #label, reg1, reg2`	`mov #label, reg2` `or reg1, reg2`
`ori label, reg1, reg2`	`mov label, reg2` `or reg1, reg2`
`ori $label, reg1, reg2`	`mov $label, reg2` `or reg1, reg2`

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

xor

Exclusive OR.

[Syntax]

- xor reg1, reg2
- xor imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "xor reg1, reg2"
  Exclusive-ORs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.
- Syntax "xor imm, reg2"
  Exclusive-ORs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operan.

[Description]

- When this instruction is executed in syntax "xor reg1, reg2", the assembler generates one xor machine instruction.
- When this instruction is executed in syntax "xor imm, reg2", the assembler executes instruction expansion to generate two or more machine instructions[Note].

(a) 0

`xor 0, reg`	`xor r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

`xor imm16, reg`	`xori imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`xor imm5, reg`	`mov imm5, r1` `xor r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

`xor imm16, reg`	`movea imm16, r0, r1` `xor r1, reg`

(e) Absolute expression exceeding the above ranges
If all the lower 16 bits of the value of imm are 0

`xor imm, reg`	`movhi HIGHW(imm), r0, r1` `xor r1, reg`

Else

`xor imm, reg`	`mov imm, r1` `xor r1, reg`

(f) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`xor  $label, reg`	`movea  $label, r0, r1` `xor  r1, reg`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`xor  #label, reg`	`mov  #label, r1` `xor  r1, reg`
`xor  label, reg`	`mov  label, r1` `xor  r1, reg`
`xor  $label, reg`	`mov  $label, r1` `xor  r1, reg`

Note The xor machine instruction take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

xori

Exclusive OR (Immediate).

[Syntax]

- xori imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

Exclusive-ORs the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the third operand.

[Description]

- If the following is specified for imm, the assembler generates one xori machine instruction[Note 1].

(a) Absolute expression having a value in the range of 0 to 65,535

`xori imm16, reg1, reg2`	`xori imm16, reg1, reg2`

(b) Relative expression having !label or %label[Note 2]

`xori !label, reg1, reg2`	`xori !label, reg1, reg2`
`xori %label, reg1, reg2`	`xori %label, reg1, reg2`

(c) Expression with HIGHW, LOWW, or HIGHW1

`xori imm16, reg1, reg2`	`xori imm16, reg1, reg2`

Note 1. The xori machine instruction takes an immediate value of 0 to 65,535 (0 to 0xFFFF) as the first operand.

Note 2. Values from -32768 to 32767 are specifiable for !label and $label and handled as immediate values (padded with zeros).

- 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 in the range of -16 to -1
If reg2 is r0

`xori imm5, reg1, r0`	`mov imm5, r1` `xor reg1, r1`

Else

`xori imm5, reg1, reg2`	`mov imm5, reg2` `xor reg1, reg2`

(b) Absolute expression having a value in the range of -32,768 to -17
If reg2 is r0

`xori imm16, reg1, r0`	`movea imm16, r0, r1` `xor reg1, r1`

Else

`xori imm16, reg1, reg2`	`movea imm16, r0, reg2` `xor 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

`xori imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `xor reg1, r1`

Else

`xori imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `xor reg1, reg2`

<2> Else
If reg2 is r0

`xori imm, reg1, r0`	`mov imm, r1` `xor reg1, r1`

Else

`xori imm, reg1, reg2`	`mov imm, reg2` `xor reg1, reg2`

(d) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section
If reg2 is r0

`xori $label, reg1, r0`	`movea $label, r0, r1` `xor reg1, r1`

Else

`xori $label, reg1, reg2`	`movea $label, r0, reg2` `xor 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

`xori #label, reg1, r0`	`mov #label, r1` `xor reg1, r1`
`xori label, reg1, r0`	`mov label, r1` `xor reg1, r1`
`xori $label, reg1, r0`	`mov $label, r1` `xor reg1, r1`

Else

`xori #label, reg1, reg2`	`mov #label, reg2` `xor reg1, reg2`
`xori label, reg1, reg2`	`mov label, reg2` `xor reg1, reg2`
`xori $label, reg1, reg2`	`mov $label, reg2` `xor reg1, reg2`

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

and

Logical product.

[Syntax]

- and reg1, reg2
- and imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "and reg1, reg2"
  ANDs the value of the register specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.
- Syntax "and imm, reg2"
  ANDs the value of the absolute expression or relative expression specified by the first operand with the value of the register specified by the second operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax "and reg1, reg2", the assembler generates one and machine instruction.
- When this instruction is executed in syntax "and imm, reg2", the assembler executes instruction expansion to generate one or more machine instruction[Note].

(a) 0

`and 0, reg`	`and r0, reg`

(b) Absolute expression having a value in the range of 1 to 65,535

`and imm16, reg`	`andi imm16, reg, reg`

(c) Absolute expression having a value in the range of -16 to -1

`and imm5, reg`	`mov imm5, r1` `and r1, reg`

(d) Absolute expression having a value in the range of -32,768 to -17

`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

`and imm, reg`	`movhi HIGHW(imm), r0, r1` `and r1, reg`

Else

`and imm, reg`	`mov imm, r1` `and r1, reg`

(f) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`and  $label, reg`	`movea  $label, r0, r1` `and  r1, reg`

(g) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`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`

Note The and machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

andi

Logical product (immediate).

[Syntax]

- andi imm, reg1, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

[Function]

ANDs the value of the absolute expression, relative expression, or expression with HIGHW, LOWW, or HIGHW1 applied specified by the first operand with the value of the register specified by the second operand, and stores the result into the register specified by the third operand.

[Description]

- If the following is specified as imm, the assembler generates one andi machine instruction[Note 1].

(a) Absolute expression having a value in the range of 0 to 65,535

`andi imm16, reg1, reg2`	`andi imm16, reg1, reg2`

(b) Relative expression having !label or %label[Note 2]

`andi !label, reg1, reg2`	`andi !label, reg1, reg2`
`andi %label, reg1, reg2`	`andi %label, reg1, reg2`

(c) Expression with HIGHW, LOWW, or HIGHW1

`andi imm16, reg1, reg2`	`andi imm16, reg1, reg2`

Note 1. The andi machine instruction takes an immediate value of 0 to 65,535 (0 to 0xFFFF) as the first operand.

Note 2. Values from -32768 to 32767 are specifiable for !label and $label and handled as immediate values (padded with zeros).

- 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 in the range of -16 to -1
If reg2 is r0

`andi imm5, reg1, r0`	`mov imm5, r1` `and reg1, r1`

Else

`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

`andi imm16, reg1, r0`	`movea imm16, r0, r1` `and reg1, r1`

Else

`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

`andi imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `and reg1, r1`

Else

`andi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `and reg1, reg2`

<2> Else
If reg2 is r0

`andi imm, reg1, r0`	`mov imm, r1` `and reg1, r1`

Else

`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

`andi $label, reg1, r0`	`movea $label, r0, r1` `and reg1, r1`

Else

`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

`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

`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`

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

not

Logical negation (takes 1's complement).

[Syntax]

- not reg1, reg2
- not imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "not reg1, reg2"
  NOTs (1's complement) the value of the register specified by the first operand, and stores the result in the register specified by the second operand.
- Syntax "not imm, reg2"
  NOTs (1's complement) the value of the absolute expression or relative expression specified by the first operand, and stores the result in the register specified by the second operand.

[Description]

- When this instruction is executed in syntax "not reg1, reg2", the assembler generates one not machine instruction.
- When this instruction is executed in syntax "not imm, reg2", the assembler executes instruction expansion to generate one or more machine instructions[Note].

(a) 0

`not 0, reg`	`not r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`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

`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

`not imm, reg`	`movhi HIGHW(imm), r0, r1` `not r1, reg`

Else

`not imm, reg`	`mov imm, r1` `not r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`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

`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`

Note The not machine instruction does not take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

shr

Logical right shift.

[Syntax]

- shr reg1, reg2
- shr imm5, reg2
- shr reg1, reg2, reg3

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "shr reg1, reg2"
  Logically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "shr imm5, reg2"
  Logically shifts to the right the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "shr reg1, reg2, reg3"
  Logically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

The assembler generates one shr machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified as imm5 in syntax "shr imm5, reg2", the assembler outputs the following message, and continues assembling by using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

Note The shr machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1F) as the first operand.

sar

Arithmetic right shift.

[Syntax]

- sar reg1, reg2
- sar imm5, reg2
- sar reg1, reg2, reg3

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "sar reg1, reg2"
  Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "sar imm5, reg2"
  Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "sar reg1, reg2, reg3"
  Arithmetically shifts to the right the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

The assembler generates one sar machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified for imm5 in syntax "sar imm5, reg2", the assembler outputs the following message, and continues assembling using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

Note The sar machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1F) as the first operand.

shl

Logical left shift.

[Syntax]

- shl reg1, reg2
- shl imm5, reg2
- shl reg1, reg2, reg3

The following can be specified for imm5:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "shl reg1, reg2"
  Logically shifts to the left the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "shl imm5, reg2"
  Logically shifts to the left the value of the register specified by the second operand by the number of bits specified by the value of the absolute expression specified by the first operand, then stores the result in the register specified by the second operand.
- Syntax "shl reg1, reg2, reg3"
  Logically shifts to the left the value of the register specified by the second operand by the number of bits indicated by the lower 5 bits of the register value specified by the first operand, then stores the result in the register specified by the third operand.

[Description]

The assembler generates one shl machine instruction.

[Flag]

CY	1 if the value of the bit shifted out last is 1, 0 if not (0 if the specified number of bits is 0)
OV	0
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified for imm5 in syntax "shl imm5, reg2", the assembler outputs the following message, and continues assembling by using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

Note The shl machine instruction takes an immediate value of 0 to 31 (0x0 to 0x1F) as the first operand.

sxb

Sign extension of byte data.

[Syntax]

- sxb reg

[Function]

Sign-extends the data of the lowermost byte of the register specified by the first operand to word length.

[Description]

The assembler generates one sxb machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

sxh

Sign extension of 2-byte data.

[Syntax]

- sxh reg

[Function]

Sign-extends the data of the lower 2 bytes of the register specified by the first operand to word length.

[Description]

The assembler generates one sxh machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

zxb

Zero extension of byte data.

[Syntax]

- zxb reg

[Function]

Zero-extends the data of the lowermost byte of the register specified by the first operand to word length.

[Description]

The assembler generates one zxb machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

zxh

Zero extension of 2-byte data

[Syntax]

- zxh reg

[Function]

Zero-extends the data of the lower 2 bytes of the register specified by the first operand to word length.

[Description]

The assembler generates one zxh machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

bsh

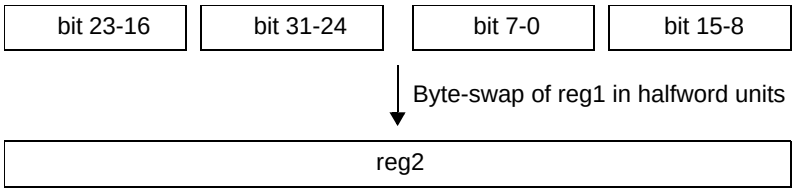
Byte swap of half-word data.

[Syntax]

- bsh reg1, reg2

[Function]

Byte-swaps the register value specified by the first operand in halfword units and stores the result in the register specified by the second operand.



[Description]

The assembler generates one bsh machine instruction.

[Flag]

CY	1 if either or both of the bytes in the lower halfword of the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the lower half-word data of the result is 0, 0 if not
SAT	---

bsw

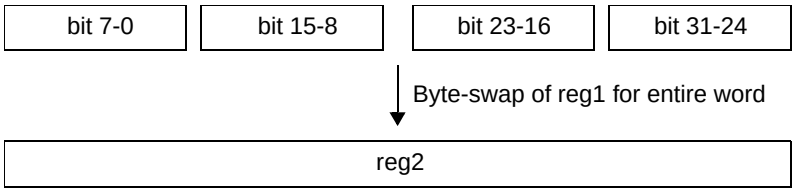
Byte swap of word data.

[Syntax]

- bsw reg1, reg2

[Function]

Byte-swaps the register value specified by the first operand and stores the result in the register specified by the second operand.



[Description]

The assembler generates one bsw machine instruction.

[Flag]

CY	1 if one or more bytes of the word in the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the word data of the result is 1, 0 if not
SAT	---

hsh

Half-word swap of half-word data.

[Syntax]

- hsh reg1, reg2

[Function]

Stores the register value specified by the first operand in the register specified by the second operand, and stores the flag assessment result in the PSW register.

[Description]

The assembler generates one hsh machine instruction.

[Flag]

CY	1 if the lower half-word data of the result is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the lower half-word data of the result is 0, 0 if not
SAT	---

hsw

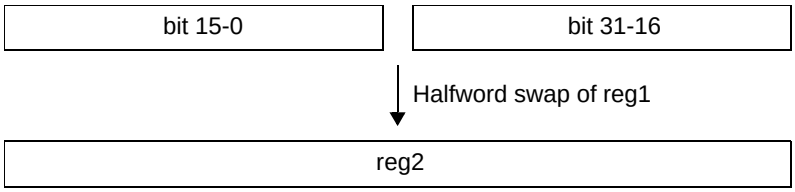
Half-word swap of word data.

[Syntax]

- hsw reg1, reg2

[Function]

Halfword-swaps the register value specified by the first operand and stores the result in the register specified by the second operand.



[Description]

The assembler generates one hsw machine instruction.

[Flag]

CY	1 if one or more halfwords in the word of the register is 0, 0 if not
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the word data of the result is 1, 0 if not
SAT	---

tst

Test.

[Syntax]

- tst reg1, reg2
- tst imm, reg2

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits
- Relative expression

[Function]

- Syntax "tst reg1, reg2"
  ANDs the value of the register specified by the second operand with the value of the register specified by the first operand, and sets only the flags without storing the result.
- Syntax "tst imm, reg2"
  ANDs the value of the register specified by the second operand with the value of the absolute expression or relative expression specified by the first operand, and sets only the flags without storing the result.

[Description]

- When this instruction is executed in syntax "tst reg1, reg2", the assembler generates one tst machine instruction.
- When this instruction is executed in syntax "tst imm, reg2", the assembler executes instruction expansion to generate two or more machine instructions[Note].

(a) 0

`tst 0, reg`	`tst r0, reg`

(b) Absolute expression having a value of other than 0 within the range of -16 to +15

`tst imm5, reg`	`mov imm5, r1` `tst r1, reg`

(c) Absolute expression exceeding the range of -16 to +15, but within the range of -32,768 to +32,767

`tst imm16, reg`	`movea imm16, r0, r1` `tst 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

`tst imm, reg`	`movhi HIGHW(imm), r0, r1` `tst r1, reg`

Else

`tst imm, reg`	`mov imm, r1` `tst r1, reg`

(e) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`tst $label, reg`	`movea $label, r0, r1` `tst 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

`tst #label, reg`	`mov #label, r1` `tst r1, reg`
`tst label, reg`	`mov label, r1` `tst r1, reg`
`tst $label, reg`	`mov $label, r1` `tst r1, reg`

Note The tst machine instruction take an immediate value as an operand.

[Flag]

CY	---
OV	0
S	1 if the word data MSB of the result is 1, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

sch0l

Bit (0) search from MSB side (search zero from left).

[Syntax]

- sch0l reg1, reg2

[Function]

Searches the word data of the register specified by the first operand, from the left (MSB side), and stores the position of the first bit (0) found in the register specified by the second operand in hexadecimal. (For example, if bit 31 of the register specified by the first operand is 0, 01H is stored in the register specified by the second operand.)

If no bit (0) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

The assembler generates one sch0l machine instruction.

[Flag]

CY	1 if a bit (0) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (0) is not found, 0 if not
SAT	---

sch0r

Bit (0) search from LSB side (search zero from right).

[Syntax]

- sch0r reg1, reg2

[Function]

Searches the word data of the register specified by the first operand, from the right (LSB side), and stores the position of the first bit (0) found in the register specified by the second operand in hexadecimal. (For example, if bit 0 of the register specified by the first operand is 0, 01H is stored in the register specified by the second operand.)

If no bit (0) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

The assembler generates one sch0r machine instruction.

[Flag]

CY	1 if a bit (0) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (0) is not found, 0 if not
SAT	---

sch1l

Bit (1) search from MSB side (search one from left).

[Syntax]

- sch1l reg1, reg2

[Function]

Searches the word data of the register specified by the first operand, from the left (MSB side), and stores the position of the first bit (1) found in the register specified by the second operand in hexadecimal. (For example, if bit 31 of the register specified by the first operand is 1, 01H is stored in the register specified by the second operand.)

If no bit (1) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (0) is found at the end, the CY flag is set (1).

[Description]

The assembler generates one sch1l machine instruction.

[Flag]

CY	1 if a bit (1) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (1) is not found, 0 if not
SAT	---

sch1r

Bit (1) search from LSB side (search zero from right).

[Syntax]

- sch1r reg1, reg2

[Function]

Searches the word data of the register specified by the first operand, from the right (LSB side), and stores the position of the first bit (1) found in the register specified by the second operand in hexadecimal. (For example, if bit 0 of the register specified by the first operand is 1, 01H is stored in the register specified by the second operand.)

If no bit (1) is found, 0 is written into the register specified by the second operand, and the Z flag is simultaneously set (1). If a bit (1) is found at the end, the CY flag is set (1).

[Description]

The assembler generates one sch1r machine instruction.

[Flag]

CY	1 if a bit (1) is found at the end, 0 if not
OV	0
S	0
Z	1 if a bit (1) is not found, 0 if not
SAT	---

5.9.5 Branch instructions

This section describes the branch instructions. Next table lists the instructions described in this section.
See the RH850 product user's manual and architecture edition for details.

Table 5.35 Branch Instructions

Instruction	Meanings
jmp	Unconditional branch
jmp32	Unconditional branch
jr	Unconditional branch (PC relative)
jr22	Unconditional branch (PC relative)
jr32	Unconditional branch (PC relative)
jcnd	Conditional branch
jcnd9	Conditional branch
jcnd17	Conditional branch
jarl	Jump and register link
jarl22	Jump and register link
jarl32	Jump and register link

jmp

Unconditional branch.

[Syntax]

- jmp [reg]
- jmp addr
- jmp disp32[reg]

The following can be specified for addr:

- Relative expression having the absolute address reference of a label

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression with a reference to the absolute address of a label and a reference to the offset within a section

[Function]

- Syntax "jmp [reg]"
  Transfers control to the address indicated by the value of the register specified by the operand.
- Syntax "jmp disp32[reg]"
  Transfers control to the address attained by adding the displacement specified by the operand and the register content.
- Syntax "jmp addr"
  Transfers control to the address indicated by the value of the relative expression specified by the operand.

[Description]

- When this instruction is executed in syntax "jmp [reg]", the assembler generates one jmp machine instruction.
- If the instruction is executed in syntax "jmp addr", when the RH850 operate, the assembler generates one jmp machine instruction (6-byte long instruction).
- When this instruction is executed in syntax "jmp disp32[reg]", the assembler generates one jmp (6-byte long instruction) machine instructions.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an expression other than a relative expression having the absolute address reference of a label is specified as addr in syntax "jmp addr", the assembler outputs the following message and stops assembling.

E0550224 : Illegal operand (label reference for jmp must be string).

jmp32

Unconditional branch.

[Syntax]

- jmp32 disp32[reg]
- jmp32 addr

The following can be specified for addr:

- Relative expression having the absolute address reference of a label

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression with a reference to the absolute address of a label and a reference to the offset within a section

[Function]

- Syntax "jmp32 disp32[reg]"
  Transfers control to the address attained by adding the displacement specified by the operand and the register content.
- Syntax "jmp32 addr"
  Transfers control to the address indicated by the value of the relative expression specified by the operand.

[Description]

The assembler generates one jmp machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an expression other than a relative expression having the absolute address reference of a label is specified as addr in syntax "jmp32 addr", the assembler outputs the following message and stops assembling.

```
E0550224 : Illegal operand (label reference for jmp must be string).
```

jr

Unconditional branch (PC relative).

[Syntax]

- jr disp22
- jr disp32

The following can be specified for disp22:

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression having a PC offset reference of label

[Function]

- Syntax "jr disp22"
  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.
- Syntax "jr disp32"
  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

[Description]

- If the instruction is executed in syntax "jr disp22", the assembler generates one jr machine instruction[Note] if any of the following expressions are specified for disp22.
  (a) Absolute expression having a value in the range of -2,097,152 to +2,097,151
  (b) Relative expression that has a PC offset reference of label having a definition in the same section of the same file as this instruction, and having a value in the range of -2,097,152 to +2,097,151
  (c) Relative expression having a PC offset reference of a label with no definition in the same file or section as this instruction

Note The jr machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xFE00000 to 0x1FFFFF) as the displacement.

- If the instruction is executed in syntax "jr disp32", the assembler generates one jr machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -2,097,152 to +2,097,151, is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp22/disp32, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

- When the -Xfar_jump assembler option is not specified, and an absolute expression outside of the range -2,097,152 to +2,097,151 or a relative expression outside of the range -2,097,152 to +2,097,151, having a label PC offset reference with a definition in the same file and same section as this instruction, is specified as disp32, the following message is output and assembly is stopped.

```
E0550230 : illegal operand (range error in displacement)
```

jr22

Unconditional branch (PC relative).

[Syntax]

- jr22 disp22

The following can be specified for disp22:

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the operand.

[Description]

- If the following is specified for disp22, the assembler generates one jr machine instruction[Note].
  - (a) Absolute value in the range of -2,097,152 to +2,097,151
  - (b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151
  - (c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jr machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xFE00000 to 0x1FFFFF) as the displacement.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22, the assembler outputs the following message and stops assembling.

E0550226 : illegal operand (must be even displacement)

jr32

Unconditional branch (PC relative).

[Syntax]

- jr32 disp32

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand.

[Description]

The assembler generates one jr machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp32, the assembler outputs the following message and stops assembling.

```
E0550226: illegal operand (must be even displacement)
```

jcnd

Conditional branch.

[Syntax]

- *jcnd* disp22

The following can be specified for disp22:

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Compares the flag condition indicated by string cnd (see "Table 5.36 jcnd Instruction List") with the current flag condition. If they are found to be the same, transfers control to the address obtained by adding the value of the absolute expression or relative expression specified by the operand to the current value of the program counter (PC)[Note].

Note For a *jcnd* instruction other than jbr, the mnemonic "b*cnd*" can be used, and the "br" machine-language instruction can be used for the jbr instruction (there is no functional difference).

Table 5.36 *jcnd* Instruction List

Instruction	Flag Condition	Meaning of Flag Condition
jgt	((S xor OV) or Z) = 0	Greater than (signed)
jge	(S xor OV) = 0	Greater than or equal (signed)
jlt	(S xor OV) = 1	Less than (signed)
jle	((S xor OV) or Z) = 1	Less than or equal (signed)
jh	(CY or Z) = 0	Higher (Greater than)
jnl	CY = 0	Not lower (Greater than or equal)
jl	CY = 1	Lower (Less than)
jnh	(CY or Z) = 1	Not higher (Less than or equal)
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	---	Always (Unconditional)
jsa	SAT = 1	Saturated

[Description]

- If the following is specified for disp22, the assembler generates one bcond machine instruction[Note].
  - (a) Absolute expression having a value in the range of -256 to +255
  - (b) Absolute expression having a PC offset reference for a label with a definition in the same section and the same file as this instruction and having a value in the range of -256 to +255

`jcnd disp9`	`bcnd disp9`

Note The b*cnd* machine instruction takes an immediate value in the range of -256 to +255 (0xFFFFFF00 to 0xFF) as the displacement.

- If the following is specified as disp22, the assembler executes instruction expansion and generates two or more machine instructions.
  - (a) Absolute expression having a value exceeding the range of -256 to +255 but within the range of -2,097,150 to +2,097,153[Note 1]
  - (b) 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 -2,097,150 to +2,097,153
  - (c) Relative expression having a PC offset reference of label without a definition in the same file or section as this instruction

`jbr disp22`	`jr disp22`

`jsa disp22`	`bsa Label1`
	`br Label2`
	`Label1:`
	`jr disp22 - 4`
	`Label2:`

`jcnd disp22`	`bncnd Label`[Note 2]
	`jr disp22 - 2`
	`Label:`

Note 1. The range of -2,097,150 to +2,097,153 applies to instructions other than jbr and jsa. The range for the jbr instruction is from -2,097,152 to +2,097,151, and that for the jsa instruction is from -2,097,148 to +2,097,155.

Note 2. bn*cnd* denotes an instruction that effects control branches under opposite conditions, for example, bnz for bz or ble for bgt.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -2,097,150 to +2,097,153, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -2,097,150 to +2,097,153, is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

- If you wish to use a j*cnd* instruction with 17-bit absolute expression as disp22, use the j*cnd*17 instruction instead.

*jcnd*9

Conditional branch.

[Syntax]

- *jcnd*9 disp9

The following can be specified for disp9:

- Absolute expression having a value of up to 17 bits
- Relative expression having a PC offset reference of label

[Function]

Compares the flag condition indicated by string cnd (see "Table 5.37 jcnd9 Instruction List") with the current flag condition. If they are found to be the same, transfers control to the address obtained by adding the value of the absolute expression or relative expression specified by the operand to the current value of the program counter (PC)[Note].

Note For a *jcnd* instruction other than jbr, the mnemonic "b*cnd*" can be used, and the "br" machine-language instruction can be used for the jbr instruction (there is no functional difference).

Table 5.37 *jcnd*9 Instruction List

Instruction	Flag Condition	Meaning of Flag Condition
jgt	((S xor OV) or Z) = 0	Greater than (signed)
jge	(S xor OV) = 0	Greater than or equal (signed)
jlt	(S xor OV) = 1	Less than (signed)
jle	((S xor OV) or Z) = 1	Less than or equal (signed)
jh	(CY or Z) = 0	Higher (Greater than)
jnl	CY = 0	Not lower (Greater than or equal)
jl	CY = 1	Lower (Less than)
jnh	(CY or Z) = 1	Not higher (Less than or equal)
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	---	Always (Unconditional)
jsa	SAT = 1	Saturated

[Description]

- If the following is specified for disp9, the assembler generates one bcond machine instruction.
  - (a) Absolute expression having a value in the range of -256 to +255
  - (b) Absolute expression having a PC offset reference for a label with a definition in the same section and the same file as this instruction and having a value in the range of -256 to +255

`jcnd disp9`	`bcnd disp9`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -256 to +255, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -256 to +255, is specified as disp9, the assembler outputs the following message and stops assembling.

E0550230 : illegal operand (range error in displacement)

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp9, the assembler outputs the following message and stops assembling.

E0550226 : illegal operand (must be even displacement)

*jcnd*17

Conditional branch.

[Syntax]

- *jcnd*17 disp17

The following can be specified for disp17:

- Absolute expression having a value of up to 9 bits
- Relative expression having a PC offset reference of label

[Function]

Compares the flag condition indicated by string cnd (see "Table 5.38 jcnd17 Instruction List") with the current flag condition. If they are found to be the same, transfers control to the address obtained by adding the value of the absolute expression or relative expression specified by the operand to the current value of the program counter (PC)[Note].

Note For a *jcnd* instruction other than jbr, the mnemonic "b*cnd*" can be used, and the "br" machine-language instruction can be used for the jbr instruction (there is no functional difference).

Table 5.38 *jcnd*17 Instruction List

Instruction	Flag Condition	Meaning of Flag Condition
jgt	((S xor OV) or Z) = 0	Greater than (signed)
jge	(S xor OV) = 0	Greater than or equal (signed)
jlt	(S xor OV) = 1	Less than (signed)
jle	((S xor OV) or Z) = 1	Less than or equal (signed)
jh	(CY or Z) = 0	Higher (Greater than)
jnl	CY = 0	Not lower (Greater than or equal)
jl	CY = 1	Lower (Less than)
jnh	(CY or Z) = 1	Not higher (Less than or equal)
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	---	Always (Unconditional)
jsa	SAT = 1	Saturated

[Description]

- If the following is specified for disp17, the assembler generates one bcond machine instruction.
  - (a) Absolute expression having a value in the range of -65,536 to +65,535
  - (b) Absolute expression having a PC offset reference for a label with a definition in the same section and the same file as this instruction and having a value in the range of -65,536 to +65,535

`jcnd disp17`	`bcnd disp17`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of -65,536 to +65,536, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having a value exceeding the range of -65,536 to +65,535, is specified as disp17, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp17, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

jarl

Jump and register link.

[Syntax]

- jarl disp22, reg2
- jarl disp32, reg2
- jarl [reg1], reg3

The following can be specified for disp22:

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression having a PC offset reference of label

[Function]

- Syntax "jarl disp22, reg2"
  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand. The return address is stored in the register specified by the second operand.
- Syntax "jarl disp32, reg2"
  Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand. The return address is stored in the register specified by the second operand.
- Syntax "jarl [reg1], reg3"
  This moves control to the address indicated by the register value specified in the first operand, plus the value of the current program counter (PC). The return address is stored in the register specified in the second operand.

[Description]

- If the instruction is executed in syntax "jarl disp22, reg2", the assembler generates one jarl machine instruction[Note] if any of the following expressions are specified for disp22.
  (a) Absolute value in the range of -2,097,152 to +2,097,151
  (b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151
  (c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jarl machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xFE00000 to 0x1FFFFF) as the operand.

- If the instruction is executed in syntax "jarl disp32, reg2", the assembler generates one jarl machine instruction (6-byte long instruction).
- If the instruction is executed in syntax "jarl [reg1], reg3", the assembler generates one jarl machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22/disp32, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

- When the -Xfar_jump assembler option is not specified, and an absolute expression outside of the range -2,097,152 to +2,097,151 or a relative expression outside of the range -2,097,152 to +2,097,151, having a label PC offset reference with a definition in the same file and same section as this instruction, is specified as disp32, the following message is output and assembly is stopped.

```
E0550230 : illegal operand (range error in displacement)
```

- If r0 is specified as reg3, the assembler outputs the following message and stops assembling.

```
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

jarl22

Jump and register link.

[Syntax]

- jarl22 disp22, reg1

The following can be specified for disp22:

- Absolute expression having a value of up to 22 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand. The return address is stored in the register specified by the second operand.

[Description]

- If the following is specified for disp22, the assembler generates one jarl machine instruction[Note].
  - (a) Absolute value in the range of -2,097,152 to +2,097,15
  - (b) Relative expression that has a PC offset reference of label having a definition in the same section and the same file as this instruction, and which has a value in the range of -2,097,152 to +2,097,151
  - (c) Relative expression having a PC offset reference of a label having no definition in the same file or section as this instruction

Note The jarl machine instruction takes an immediate value in the range of -2,097,152 to +2,097,151 (0xFE00000 to 0x1FFFFF) as the operand.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression that exceeds the range of -2,097,152 to +2,097,151, or a relative expression having a PC offset reference of label with a definition in the same section and the same file as this instruction and having a value that falls outside the range of -2,097,152 to +2,097,151 is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550230 : illegal operand (range error in displacement)
```

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction and having an odd-numbered value, is specified as disp22, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

jarl32

Jump and register link.

[Syntax]

- jarl32 disp32, reg2

The following can be specified for disp32:

- Absolute expression having a value of up to 32 bits
- Relative expression having a PC offset reference of label

[Function]

Transfers control to the address attained by adding the current program counter (PC) value and the relative or absolute expression value specified by the first operand. The return address is stored in the register specified by the second operand.

[Description]

The assembler generates one jarl machine instruction (6-byte long instruction).

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having an odd-numbered value, or a relative expression having a PC offset reference of a label with a definition in the same section and the same file as this instruction, and having an odd-numbered value, is specified as disp32, the assembler outputs the following message and stops assembling.

```
E0550226 : illegal operand (must be even displacement)
```

5.9.6 Bit manipulation instructions

This section describes the bit manipulation instructions. Next table lists the instructions described in this section.
See the RH850 product user's manual and architecture edition for details.

Table 5.39 Bit Manipulation Instructions

Instruction	Meanings
set1	Sets bit
clr1	Clears bit
not1	Inverts bit
tst1	Tests bit

set1

Set s bit.

[Syntax]

- set1 bit#3, disp[reg1]
- set1 reg2, [reg1]

The following can be specified for disp:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

Caution The disp cannot be specified in syntax "set1 reg2, [reg1]".

[Function]

- Syntax "set1 bit#3, disp[reg1]"
  Sets the bit specified by the first operand of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.
- Syntax "set1 reg2, [reg1]"
  Sets the bit specified by the register value specified by the first operand of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

[Description]

- If the following is specified for disp, the assembler generates one set1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`set1 bit#3, disp16[reg1]`	`set1 bit#3, disp16[reg1]`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`set1 bit#3, $label[reg1]`	`set1 bit#3, $label[reg1]`

(c) Relative expression having !label or %label

`set1 bit#3, !label[reg1]`	`set1 bit#3, !label[reg1]`
`set1 bit#3, %label[reg1]`	`set1 bit#3, %label[reg1]`

(d) Expression with HIGHW, LOWW, or HIGHW1

`set1 bit#3, HIGHW(value)[reg1]`	`set1 bit#3, HIGHW(value)[reg1]`
`set1 bit#3, LOWW(value)[reg1]`	`set1 bit#3, LOWW(value)[reg1]`
`set1 bit#3, HIGHW1(value)[reg1]`	`set1 bit#3, HIGHW1(value)[reg1]`

Note The set1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement.

- If any of the following is specified as disp, the assembler executes instruction expansion to generate two or more machine instructions.

  (a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`set1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `set1 bit#3, LOWW(disp)[r1]`

  (b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`set1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `set1 bit#3, LOWW(#label)[r1]`
`set1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `set1 bit#3, LOWW(label)[r1]`
`set1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `set1 bit#3, LOWW($label)[r1]`

- If disp is omitted, the assembler assumes 0.
- If a relative expression with #label, or a relative expression with #label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression with $label, or a relative expression with $label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

clr1

Clears bit.

[Syntax]

- clr1 bit#3, disp[reg1]
- clr1 reg2, [reg1]

The following can be specified for disp:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

Caution The disp cannot be specified in syntax "clr1 reg2, [reg1]".

[Function]

- Syntax "clr1 bit#3, disp[reg1]"
  Clears the bit specified by the first operand of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.
- Syntax "clr1 reg2, [reg1]"
  Clears the bit specified by the register value specified by the first operand of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

[Description]

- If the following is specified as disp, the assembler generates one clr1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`clr1 bit#3, disp16[reg1]`	`clr1 bit#3, disp16[reg1]`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`clr1 bit#3, $label[reg1]`	`clr1 bit#3, $label[reg1]`

(c) Relative expression having !label or %label

`clr1 bit#3, !label[reg1]`	`clr1 bit#3, !label[reg1]`
`clr1 bit#3, %label[reg1]`	`clr1 bit#3, %label[reg1]`

(d) Expression with HIGHW, LOWW, or HIGHW1

`clr1 bit#3, HIGHW(value)[reg1]`	`clr1 bit#3, HIGHW(value)[reg1]`
`clr1 bit#3, LOWW(value)[reg1]`	`clr1 bit#3, LOWW(value)[reg1]`
`clr1 bit#3, HIGHW1(value)[reg1]`	`clr1 bit#3, HIGHW1(value)[reg1]`

Note The clr1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement.

- If any of the following is specified as disp, the assembler executes instruction expansion to generate two or more machine instructions.

  (a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`clr1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `clr1 bit#3, LOWW(disp)[r1]`

  (b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`clr1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `clr1 bit#3, LOWW(#label)[r1]`
`clr1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `clr1 bit#3, LOWW(label)[r1]`
`clr1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `clr1 bit#3, LOWW($label)[r1]`

- If disp is omitted, the assembler assumes 0.
- If a relative expression with #label or a relative expression with #label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] that follows the expression can be omitted. If omitted, the assembler assumes [r0] to be specified.
- If a relative expression with $label, or a relative expression with $label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

not1

Inverts bit.

[Syntax]

- not1 bit#3, disp[reg1]
- not1 reg2, [reg1]

The following can be specified for disp:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

Caution The disp cannot be specified in syntax "not1 reg2, [reg1]".

[Function]

- Syntax "not1 bit#3, disp[reg1]"
  Inverts the bit specified by the first operand (0 to 1 or 1 to 0) of the data indicated by the address specified by the second operand. The bits other than the one specified are not affected.
- Syntax "not1 reg2, [reg1]"
  Inverts the bit specified by the register value specified by the first operand (0 to 1 or 1 to 0) of the data indicated by the address specified by the register value of the second operand. The bits other than the one specified are not affected.

[Description]

- If the following is specified for disp, the assembler generates one not1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`not1 bit#3, disp16[reg1]`	`not1 bit#3, disp16[reg1]`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`not1 bit#3, $label[reg1]`	`not1 bit#3, $label[reg1]`

(c) Relative expression having !label or %label

`not1 bit#3, !label[reg1]`	`not1 bit#3, !label[reg1]`
`not1 bit#3, %label[reg1]`	`not1 bit#3, %label[reg1]`

(d) Expression with HIGHW, LOWW, or HIGHW1

`not1 bit#3, HIGHW(value)[reg1]`	`not1 bit#3, HIGHW(value)[reg1]`
`not1 bit#3, LOWW(value)[reg1]`	`not1 bit#3, LOWW(value)[reg1]`
`not1 bit#3, HIGHW1(value)[reg1]`	`not1 bit#3, HIGHW1(value)[reg1]`

Note The not1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement.

- If any of the following is specified as disp, the assembler executes instruction expansion to generate two or more machine instructions.

  (a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`not1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `not1 bit#3, LOWW(disp)[r1]`

  (b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`not1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `not1 bit#3, LOWW(#label)[r1]`
`not1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `not1 bit#3, LOWW(label)[r1]`
`not1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `not1 bit#3, LOWW($label)[r1]`

- If disp is omitted, the assembler assumes 0.
- If a relative expression with #label, or a relative expression with #label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression with $label, or a relative expression with $label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not[Note]
SAT	---

Note The flag values shown here are those existing prior to the execution of this instruction, not those after the execution.

tst1

Tests bit.

[Syntax]

- tst1 bit#3, disp[reg1]
- tst1 reg2, [reg1]

The following can be specified for disp:

- Absolute expression having a value of up to 32 bits
- Relative expression
- Either of the above expressions with HIGHW, LOWW, or HIGHW1 applied

Caution The disp cannot be specified in syntax "tst1 reg2, [reg1]".

[Function]

- Syntax "tst1 bit#3, disp[reg1]"
  Sets only a flag according to the value of the bit specified by the first operand of the data indicated by the address specified by the second operand. The value of the second operand and the specified bit are not changed.
- Syntax "tst1 reg2, [reg1]"
  Sets only a flag according to the value of the bit of the register value specified by the first operand of the data indicated by the address specified by the second operand. The value of the second operand and the specified bit are not changed.

[Description]

- If the following is specified for disp, the assembler generates one tst1 machine instruction[Note].

(a) Absolute expression having a value in the range of -32,768 to +32,767

`tst1 bit#3, disp16[reg1]`	`tst1 bit#3, disp16[reg1]`

(b) Relative expression having $label for a label having a definition in the sdata/sbss-attribute section

`tst1 bit#3, $label[reg1]`	`tst1 bit#3, $label[reg1]`

(c) Relative expression having !label or %label

`tst1 bit#3, !label[reg1]`	`tst1 bit#3, !label[reg1]`
`tst1 bit#3, %label[reg1]`	`tst1 bit#3, %label[reg1]`

(d) Expression with HIGHW, LOWW, or HIGHW1

`tst1 bit#3, HIGHW(value)[reg1]`	`tst1 bit#3, HIGHW(value)[reg1]`
`tst1 bit#3, LOWW(value)[reg1]`	`tst1 bit#3, LOWW(value)[reg1]`
`tst1 bit#3, HIGHW1(value)[reg1]`	`tst1 bit#3, HIGHW1(value)[reg1]`

Note The tst1 machine instruction takes an immediate value in the range of -32,768 to +32,767 (0xFFFF8000 to 0x7FFF) as the displacement.

- If any of the following is specified as disp, the assembler executes instruction expansion to generate two or more machine instructions.

  (a) Absolute expression having a value exceeding the range of -32,768 to +32,767

`tst1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `tst1 bit#3, LOWW(disp)[r1]`

  (b) Relative expression having #label or label, or that having $label for a label having no definition in the sdata/sbss-attribute section

`tst1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `tst1 bit#3, LOWW(#label)[r1]`
`tst1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `tst1 bit#3, LOWW(label)[r1]`
`tst1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `tst1 bit#3, LOWW($label)[r1]`

- If disp is omitted, the assembler assumes 0.
- If a relative expression with #label, or a relative expression with #label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [r0] is specified.
- If a relative expression with $label, or a relative expression with $label and with HIGHW, LOWW, or HIGHW1 applied is specified as disp, [reg1] can be omitted. If omitted, the assembler assumes that [gp] is specified.

[Flag]

CY	---
OV	---
S	---
Z	1 if the specified bit is 0, 0 if not
SAT	---

5.9.7 Stack manipulation instructions

This section describes the stack manipulation instructions. Next table lists the instructions described in this section. See the RH850 product user's manual and architecture edition for details.

Table 5.40 Stack Manipulation Instructions

Instruction	Meanings
push	Pushes to stack area (single register)
pushm	Pushes to stack area (multiple registers)
pop	Pops from stack area (single register)
popm	Pops from stack area (multiple registers)

push

Pushes to stack area (single register).

[Syntax]

push reg

[Function]

Pushes the value of the register specified by the operand to the stack area.

[Description]

- When the push instruction is executed, the assembler executes instruction expansion to generate two or more machine instructions.

`push reg`	`add -4, sp` `st.w reg, [sp]`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

Caution Instruction expansion is performed, and set via an add instruction.

pushm

Pushes to stack area (multiple registers).

[Syntax]

pushm reg1, reg2, ..., regN

[Function]

Pushes the values of the registers specified by the operand to the stack area. Up to 32 registers can be specified by the operand.

[Description]

- When the pushm instruction is executed, the assembler executes instruction expansion to generate two or more machine instructions.
  - When there are four or fewer registers.

`pushm reg1, reg2, ..., regN`	`add -4 * N, sp`
	`st.w regN, 4 * (N - 1)[sp]`
	`:`
	`st.w reg2, 4 * 1[sp]`
	`st.w reg1, 4 * 0[sp]`

  - When there are five or more registers.

`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]`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

Caution Instruction expansion is performed, and set via an add/addi instruction.

pop

Pops from stack area (single register).

[Syntax]

pop reg

[Function]

Pops the value of the register specified by the operand from the stack area.

[Description]

- When the pop instruction is executed, the assembler executes instruction expansion to generate two or more machine instructions.

`pop reg`	`ld.w [sp], reg` `add 4, sp`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

Caution Instruction expansion is performed, and set via an add instruction.

popm

Pops from stack area (multiple registers).

[Syntax]

popm reg1, reg2, ..., regN

[Function]

Pops the values of the registers specified by the operand from the stack area in the sequence in which the registers are specified. Up to 32 registers can be specified by the operand.

[Description]

- When the popm instruction is executed, the assembler executes instruction expansion to generate two or more machine instructions.
  - When there are three or fewer registers.

`popm reg1, reg2, ..., regN`	`ld.w 4 * 0[sp], reg1`
	`ld.w 4 * 1[sp], reg2`
	`:`
	`ld.w 4 * (N - 1)[sp], regN`
	`add 4 * N, sp`

  - When there are four or more registers.

`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`

[Flag]

CY	1 if a carry occurs from MSB (Most Significant Bit), 0 if not
OV	1 if Integer-Overflow occurs, 0 if not
S	1 if the result is negative, 0 if not
Z	1 if the result is 0, 0 if not
SAT	---

Caution Instruction expansion is performed, and set via an add/addi instruction.

5.9.8 Special instructions

This section describes the special instructions.
See the RH850 product user's manual and architecture edition for details.

Table 5.41 Special Instructions

Instruction	Meanings
ldsr	Loads to system register
stsr	Stores contents of system register
ldl.w	Load to start atomic word data manipulation
stc.w	Conditional storage when atomic word data manipulation is complete
cll	Link for atomic manipulation is canceled
di	Disables maskable interrupt
ei	Enables maskable interrupt
reti	Returns from trap or interrupt routine
eiret	Returns from EI level exception
feret	Returns from FE level exception
halt	Stops the processor
trap	Software trap
fetrap	FE level software exception
nop	No operation
switch	Table reference branch
callt	Table reference call
ctret	Returns from callt
caxi	Comparison and swap
rie	Reserved instruction exception
syncm	Memory synchronize instruction
syncp	Pipeline synchronize instruction
synce	Exception synchronization instruction
synci	Instruction pipeline synchronization instruction
prepare	Generates stack frame (preprocessing of function)
dispose	Deletes stack frame (post processing of function)
syscall	System call exception
pushsp	Push from the stack
popsp	Pop from the stack
snooze	Snooze

ldsr

Loads to system register.

[Syntax]

- ldsr reg, regID
- ldsr reg, regID, selID

The following can be specified as regID and selID:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "ldsr reg, regID"
  Stores the value of the register specified by the first operand in the system register[Note] indicated by the system register number specified by the second operand.
- Syntax "ldsr reg, regID, selID"
  Stores the register value specified in the first operand into the system register[Note] indicated by the system-register number specified in the second operand, and the group number specified in the third operand.
  If selID is omitted, it is assumed that selID=0 was specified.

Note For details of the system registers, see the Relevant Device's Hardware User's Manual provided with the each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

Caution If the program status word (PSW) is specified as the system register, the value of the corresponding bit of reg is set as each flag.

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified as regID, the assembler outputs the following message, then continues assembling using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate)
```

Note The ldsr machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1F) as the second operand.

- If a reserved register number, the number of a register which cannot be accessed (such as ECR) or the number of a register which can be accessed only in the debug mode is specified as regID, the assembler outputs the following message and continues assembling as is.

```
W0550018 : illegal regID for ldsr
```

stsr

Stores contents of system register.

[Syntax]

- stsr regID, reg
- stsr regID, reg, selID

The following can be specified as regID and selID:

- Absolute expression having a value of up to 5 bits

[Function]

- Syntax "stsr regID, reg"
  Stores the value of the system register[Note] indicated by the system register number specified by the first operand, to the register specified by the second operand.
- Syntax "stsr regID, reg, selID"
  Stores the value of the system register[Note] indicated by the system-register number specified in the first operand and the group number specified in the third operand into the register specified in the second operand.
  If selID is omitted, it is assumed that selID=0 was specified.

Note For details of the system registers, see the Relevant Device's Hardware User's Manual provided with the each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 31 is specified as regID and selID, the assembler outputs the following message, then continues assembling using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate)
```

Note The stsr machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1F) as the first operand.

- If a reserved register number or the number of a register which can be accessed only in the debug mode is specified as regID, the assembler outputs the following message and continues assembling as is.

```
W0550018 : illegal regID for ldsr
```

ldl.w

Load to start atomic word data manipulation (Load Linked)

[Syntax]

- ldl.w [reg1], reg2

[Function]

In order to perform an atomic read-modify-write operation, word data is read from the memory and stored in general-purpose register reg2. A link is then generated corresponding to the address range that includes the specified address.

Subsequently, if a specific condition is satisfied before an stc.w instruction is executed for this ldl.w instruction, the link will be deleted. If an stc.w instruction is executed after the link has been deleted, stc.w execution will fail.

If an stc.w instruction is executed while the link is still available, stc.w execution will succeed. The link is also deleted in this case.

The ldl.w and stc.w instructions can be used to accurately update the memory in a multi-core system.

Remark Use the ldl.w and stc.w instructions instead of the caxi instruction if an atomic guarantee is required when updating the memory in a multi-core system.

[Description]

- The assembler generates one ldl.w machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

stc.w

Conditional storage when atomic word data manipulation is complete (Store Conditional)

[Syntax]

- stc.w reg2, [reg1]

[Function]

This instruction can only be executed successfully if a link exists that corresponds to the specified address. If a corresponding link exists, the word data of general-purpose register reg2 is stored in the memory and an atomic read-modify-write is executed.

If the corresponding link has been lost, the data is not stored in the memory and execution of this instruction fails.

Whether execution of the stc.w instruction has succeeded or not can be ascertained by checking the contents of general-purpose register reg2 after the instruction has been executed.

If execution of the stc.w instruction was successful, general-purpose register reg3 will be set (1). If execution failed, reg2 will be cleared (0).

This instruction can be used together with the ldl.w instruction to ensure accurate updating of the memory in a multi-core system.

Remark Use the ldl.w and stc.w instructions instead of the caxi instruction if an atomic guarantee is required when updating the memory in a multi-core system.

[Description]

- The assembler generates one stc.w machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

cll

Link for atomic manipulation is canceled.

[Syntax]

- cll

[Function]

This instruction explicitly eliminates the link generated by the ldl.w instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---
ID	---

di

Disables maskable interrupt.

[Syntax]

- di

[Function]

Sets the ID bit of the PSW to 1 and disables acknowledgement of maskable interrupts since this instruction has already been executed.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---
ID	1

ei

Enables maskable interrupt.

[Syntax]

- ei

[Function]

Sets the ID bit of the PSW to 0, and enables acknowledgment of maskable interrupt from the next instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---
ID	0

reti

Returns from trap or interrupt routine.

[Syntax]

- reti

[Function]

Returns from a trap or interrupt routine[Note].

Note For details of the function, see the Relevant Device's Architecture User's Manual of each devic

[Flag]

CY	Extracted value
OV	Extracted value
S	Extracted value
Z	Extracted value
SAT	Extracted value

eiret

Returns from EI level exception (Return from Trap or Interrupt)

[Syntax]

- eiret

[Function]

Returns execution from an EI level exception. The return PC and PSW are loaded from the EIPC and EIPSW registers and set in the PC and PSW, and control is passed.

If EP = 0, it means that interrupt (EIINT*n*) processing has finished, so the corresponding bit of the ISPR register is cleared.

[Description]

- The assembler generates one eiret machine instruction.

[Flag]

CY	Value read from EIPSW is set
OV	Value read from EIPSW is set
S	Value read from EIPSW is set
Z	Value read from EIPSW is set
SAT	Value read from EIPSW is set

feret

Returns from FE level exception (Return from Trap or Interrupt)

[Syntax]

- feret

[Function]

Returns execution from an FE level exception. The return PC and PSW are loaded from the FEPC and FEPSW registers and set in the PC and PSW, and control is passed.

[Description]

- The assembler generates one feret machine instruction.

[Flag]

CY	Value read from FEPSW is set
OV	Value read from FEPSW is set
S	Value read from FEPSW is set
Z	Value read from FEPSW is set
SAT	Value read from FEPSW is set

halt

Stops the processor.

[Syntax]

- halt

[Function]

Stops the processor and sets it in the HALT status. The HALT status can be released by a maskable interrupt, NMI, or reset.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

trap

Software trap.

[Syntax]

- trap vector

The following can be specified for vector:

- Absolute expression having a value of up to 5 bits

[Function]

Causes a software trap[Note].

Note For details of the function, see the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value falling outside the range of 0 to 31 is specified as vector, the assembler outputs the following message, continuing assembling using the lower 5 bits[Note] of the specified value.

```
W0550011 : illegal operand (range error in immediate)
```

Note The trap machine instruction takes an immediate value in the range of 0 to 31 (0x0 to 0x1F) as an operand.

fetrap

FE level software exception (FE-level Trap)

[Syntax]

- fetrap

[Function]

Saves the contents of the return PC (address of the instruction next to the FETRAP instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores the exception cause code in the FEIC register, and updates the PSW according to the exception causes. Execution then branches to the exception handler address and exception handling is started.

Table 5.42 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset shows the correspondence between vector4 and exception cause codes and exception handler address offset. Exception handler addresses are calculated based on the offset addresses listed in Table 5.42 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset.

Table 5.42 Correspondence between vector4 and Exception Cause Codes and Exception Handler Address Offset

vector4	Exception Cause Code	Offset Address
0H	Not specifiable	
1H	00000031H	30H
2H	00000032H	
(Omission)		
FH	0000003FH	

[Description]

- The assembler generates one fetrap machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

nop

No operation.

[Syntax]

- nop

[Function]

Nothing is executed. This instruction can be used to allocate an area during an instruction sequence or to insert a delay cycle during instruction execution.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

switch

Table reference branch.

[Syntax]

switch reg

[Function]

Performs processing in the following sequence.

(1) Adds the value resulting from logically shifting the value specified by the operand 1 bit to the left to the first address of the table (address following the switch instruction) to generate a table entry address.

(2) Loads signed halfword data from the generated table entry address.

(3) Logically shifts the loaded value 1 bit to the left and sign-extends it to word length. Then adds the first address of the table to it to generate an address

(4) Branches to the generated address.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by reg, the assembler outputs the following message and stops assembling.

```
E0550239 : Illegal operand (cannot use r0 as source in RH850 mode).
```

callt

Table reference call.

[Syntax]

- callt imm6

The following can be specified as imm6:

- Absolute expression having a value of up to 6 bits

[Function]

Performs processing in the following sequence[Note]

(1) Saves the values of the return PC and PSW to CTPC and CTPSW.

(2) Generates a table entry address by shifting the value specified by the operand 1 bit to the left as an offset value from CTBP(CALLT Base Pointer) and by adding it to the CTBP value.

(3) Loads unsigned halfword data from the generated table entry address.

(4) Adds the loaded value to the CTBP value to generate an address.

(5) Branches to the generated address.

Note For details of the system registers, see the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

ctret

Returns from callt.

[Syntax]

- ctret

[Function]

Returns from the processing by callt. Performs the processing in the following sequence[Note]:

(1) Extracts the return PC and PSW from CTPC and CTPSW.

(2) Sets the extracted values in the PC and PSW and transfers control.

Note For details of the system registers, see the Relevant Device's Architecture User's Manual of each device.

[Flag]

CY	Extracted value
OV	Extracted value
S	Extracted value
Z	Extracted value
SAT	Extracted value

caxi

Comparison and swap (Compare and Exchange for Interlock)

[Syntax]

- caxi [reg1], reg2, reg3

[Function]

Word data is read from the specified address and compared with the word data in general-purpose register reg2, and the result is indicated by flags in the PSW. Comparison is performed by subtracting the read word data from the word data in general-purpose register reg2. If the comparison result is "0", word data in general-purpose register reg3 is stored in the generated address, otherwise the read word data is stored in the generated address.

Afterward, the read word data is stored in general-purpose register reg3. General-purpose registers reg1 and reg2 are not affected.

[Description]

- The assembler generates one caxi machine instruction.

[Flag]

CY	"1" if a borrow occurs in the result operation; otherwise, "0"
OV	"1" if overflow occurs in the result operation; otherwise, "0"
S	"1" if result is negative; otherwise, "0"
Z	"1" if result is 0; otherwise, "0"
SAT	---

rie

Reserved Instruction exception (Reserved Instruction Exception)

[Syntax]

- rie
- rie imm5, imm4

The following can be specified as imm5:

- Absolute expression having a value of up to 5 bits

The following can be specified as imm4:

- Absolute expression having a value of up to 4 bits

[Function]

Saves the contents of the return PC (address of the RIE instruction) and the current contents of the PSW to FEPC and FEPSW, respectively, stores the exception cause code in the FEIC register, and updates the PSW according to the exception causes. Execution then branches to the exception handler address and exception handling is started.

Exception handler addresses are calculated based on the offset address 60H.

[Description]

- The assembler generates one rie machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

syncm

Memory synchronize instruction (Synchronize Memory)

[Syntax]

- syncm

[Function]

Synchronizes the CPU execution pipeline and memory accesses.
"Synchronization" refers to the status where the result of preceding memory accesses can be referenced by any master device within the system.
In cases such as when buffering is used to delay memory accesses and synchronization of all memory accesses has not occurred, the SYNCM instruction does not complete and waits for the synchronization.
The subsequent instructions will not be executed until the SYNCM instruction execution is complete.
This instruction can be used to implement "synchronization primitives" in a multi-processing environment when the function described above is provided by the system.

[Description]

- The assembler generates one syncm machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

syncp

Pipeline synchronize instruction (Synchronize Pipline)

[Syntax]

- syncp

[Function]

Waits until execution of all previous instructions is completed before being executed.

[Description]

- The assembler generates one syncp machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

synce

Exception synchronization instruction (Synchronize Exceptions)

[Syntax]

- synce

[Function]

Waits for the synchronization of all preceding exceptions before starting execution.
It does not perform any operation but is completed when its execution is started.
"Exception synchronization" means that all exceptions that are generated by the preceding instructions are notified to the CPU and are kept waiting until their priority is judged. If a condition of acknowledging exceptions is satisfied before this instruction is executed, therefore, all imprecise exceptions (FPI exceptions) that are generated because of the preceding instructions are always acknowledged before execution of this instruction is completed.
This instruction can be used to guarantee completion of exception handling by the preceding task before a task is changed or terminated in a multi-processing environment.

[Description]

- The assembler generates one synce machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

synci

Instruction pipeline synchronization instruction (Synchronize Instruction Pipeline)

[Syntax]

- synci

[Function]

Makes subsequent instructions wait until all the instructions ahead of this instruction have finished executing. The instructions executed after the synci instruction are guaranteed to adapt to the effects produced by the execution of the instructions preceding synci. This instruction can be used to realize "self-programming code" to overwrite instructions in the memory.

Remark The synci instruction clears the CPU instruction fetch pipeline so that subsequently executed instructions are re-fetched.
If the CPU includes an instruction cache, the instruction cache must be disabled to enable the realization of this self-programming code.

[Description]

- The assembler generates one synci machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

prepare

Generates stack frame (preprocessing of function).

[Syntax]

- prepare list, imm1
- prepare list, imm1, imm2
- prepare list, imm1, sp

The following can be specified as imm1/imm2:

- Absolute expression having a value of up to 32 bits

list specifies the 12 registers that can be pushed by the prepare 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

The following two specifications are equivalent.

`prepare r26, r29, r31, 0x10`	`prepare 0x103, 0x10`

[Function]

The prepare instruction performs the preprocessing of a function.

- Syntax "prepare list, imm1"
  (a) Pushes one of the registers specified by the first operand and subtracts 4 from the stack pointer (sp).
  (b) Repeatedly performs (a) until all the registers specified by the first operand have been pushed.
  (c) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp in the register saving area.

- Syntax "prepare list, imm1, imm2"
  (a) Pushes one of the registers specified by the first operand and subtracts 4 from sp.
  (b) Repeatedly performs (a) until all the registers specified by the first operand have been pushed.
  (c) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp to the register saving area.
  (d) Sets the value of the absolute expression specified by the third operand in ep.

- Syntax "prepare list, imm1, sp"
  (a) Pushes one of the registers specified by the first operand and subtracts 4 from sp.
  (b) Repeatedly performs (a) until all the registers specified by the first operand have been pushed.
  (c) Subtracts the value of the absolute expression specified by the second operand from sp[Note] and sets sp in the register saving area.
  (d) Sets the value of sp specified by the third operand in ep.

Note Since the value actually subtracted from sp by the machine instruction is imm1 shifted 2 bits to the left, the assembler shifts the specified imm1 2 bits to the right in advance and reflects it in the code.

[Description]

- If the following is specified for imm1, the assembler generates one prepare machine instruction.

(a) Absolute expression having a value in the range of 0 to 127

`prepare list, imm1`	`prepare list, imm1`
`prepare list, imm1, imm2`	`prepare list, imm1, imm2`
`prepare list, imm1, sp`	`prepare list, imm1, sp`

- When the following is specified as imm1, the assembler executes instruction expansion to generate two or more machine instructions.

(a) Absolute expression exceeding the range of 0 to 127, but within the range of 0 to 32,767

`prepare list, imm1`	`prepare list, 0` `movea  -imm1, sp, sp`
`prepare list, imm1, imm2`	`prepare list, 0, imm2` `movea  -imm1, sp, sp`
`prepare list, imm1, sp`	`prepare list, 0, sp` `movea  -imm1, sp, sp`

(b) Absolute expression having a value exceeding the range of 0 to 32,767

`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`

`prepare list, imm1, sp`	`prepare list, 0, sp`
	`mov  imm1, r1`
	`sub  r1, sp`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

dispose

Deletes stack frame (post processing of function).

[Syntax]

- dispose imm, list
- dispose imm, list, [reg]

The following can be specified for imm:

- Absolute expression having a value of up to 32 bits

The following can be specified as list. list specifies the 12 registers that can be popped by the dispose instruction.

- Register
  Specify the registers (r20 to r31) to be popped, 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

The following two specifications are equivalent.

`dispose 0x10, r26, r29, r31`	`dispose 0x10, 0x103`

[Function]

The dispose instruction performs the postprocessing of a function.

- Syntax "dispose imm, list"
  (a) Adds the value of the absolute expression specified by the first operand to the stack pointer (sp)[Note] and sets sp in the register saving area.
  (b) Pops one of the registers specified by the second operand and adds 4 to sp.
  (c) Repeatedly executes (b) until all the registers specified by the second operand have been popped.

- Syntax "dispose imm, list, [reg]"
  (a) Adds the value of the absolute expression specified by the first operand to the stack pointer (sp)[Note] and sets sp in the register saving area.
  (b) Pops one of the registers specified by the second operand and adds 4 to sp.
  (c) Repeatedly executes (b) until all the registers specified by the second operand have been popped.
  (d) Sets the register value specified by the third operand in the program counter (PC).

Note Since the value actually added to sp by the machine instruction is imm shifted 2 bits to the left, the assembler shifts the specified imm 2 bits to the right in advance and reflects it in the code.

[Description]

- If the following is specified for imm, the assembler generates one dispose machine instruction.

  (a) Absolute expression having a value in the range of 0 to 127

`dispose imm, list`	`dispose imm, list`
`dispose imm, list, [reg]`	`dispose imm, list, [reg]`

- If the following is specified for imm, the assembler executes instruction expansion to generate two or more machine instructions.

  (a) Absolute expression exceeding the range of 0 to 127, but within the range of 0 to 32,767

`dispose imm, list`	`movea  imm, sp, sp` `dispose 0, list`
`dispose imm, list, [reg]`	`movea  imm, sp, sp` `dispose 0, list, [reg]`

  (b) Absolute expression having a value exceeding the range of 0 to 32,767

`dispose imm, list`	`mov  imm, r1`
	`add  r1, sp`
	`dispose 0, list, [reg]`

`dispose imm, list, [reg]`	`mov  imm, r1`
	`add  r1, sp`
	`dispose 0, list, [reg]`

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If r0 is specified by the [reg] in syntax "dispose imm, list, [reg]", the assembler outputs the following message and stops assembling.

E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).

syscall

System call exception (System Call)

[Syntax]

- syscall vector

The following can be specified as vector:

- Absolute expression having a value of up to 8 bits

[Function]

This instruction calls the system service of an OS.

(1) Saves the contents of the return PC (address of the instruction next to the syscall instruction) and PSW to EIPC and EIPSW.

(2) Stores the exception cause code corresponding to vector in the EIIC register.
The exception cause code is the value of vector plus 8000H.

(3) Updates the PSW according to the exception causes.

(4) Generates a 32-bit table entry address by adding the value of the SCBP register and vector that is logically shifted 2 bits to the left and zero-extended to a word length.
If vector is greater than the value specified by the SIZE bit of system register SCCFG; however, vector that is used for the above addition is handled as 0.

(5) Loads the word of the address generated in (4).

(6) Generates a 32-bit target address by adding the value of the SCBP register to the data in (5).

(7) Branches to the target address generated in (6).

[Description]

- The assembler generates one syscall machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 255 is specified as vector, the assembler outputs the following message and continues assembling by using the lower 8 bits[Note] of the specified value.

W0550011 : Illegal operand (range error in immediate).

Note The syscall machine instruction takes an immediate value in the range of 0 to 255 (0x0 to 0xFF)as the operand.

- This instruction is dedicated to calling the system service of an OS. For how to use it in the user program, see the Function Specification of each OS.
- In the syscall instruction memory read operation executed in order to read the table, memory protection is performed with the supervisor privilege.

pushsp

Push from the stack (Push Registers from Stack)

[Syntax]

- pushsp rh, rt

The following can be specified as rt and rh:

- General-purpose registers r0 - r31

[Function]

Stores general-purpose register rh to rt in the stack in ascending order (rh, rh +1, rh + 2, ..., rt). After all the specified registers have been stored, sp is updated (decremented).

Remark The lower two bits of the address specified by sp are masked by 0.
If an exception is acknowledged before sp is updated, instruction execution is halted and exception handling is executed with the start address of this instruction used as the return address. The pushsp instruction is then executed again (The sp value from before the exception handling is saved).

[Description]

- The assembler generates one pushsp machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If the relationship between the register numbers specified as rh and rt is rh > rt, the assembler outputs the following message.

```
E0550249 : Illegal syntax.
```

popsp

Pop from the stack (Pop Registers from Stack)

[Syntax]

- popsp rh, rt

The following can be specified as rt and rh:

- General-purpose registers r0 - r31

[Function]

Loads general-purpose register rt to rh from the stack in descending order (rt, rt - 1, rt - 2, ..., rh). After all the registers down to the specified register have been loaded, sp is updated (incremented).

Remark The lower two bits of the address specified by sp are masked by 0.
If an exception is acknowledged before sp is updated, instruction execution is halted and exception handling is executed with the start address of this instruction used as the return address. The popsp instruction is then executed again (The sp value from before the exception handling is saved).

[Description]

- The assembler generates one popsp machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

[Caution]

- If the relationship between the register numbers specified as rh and rt is rh > rt, the assembler outputs the following message.

```
E0550249 : Illegal syntax.
```

- If a register that includes sp (r3) is specified as the restore register (rh = 3 to 31), the value read from the memory is not stored in sp (r3). This allows the POPSP instruction to be correctly re-executed after execution has been halted.

snooze

Snooze (Snooze)

[Syntax]

- snooze

[Function]

Temporarily halts operation of the CPU for the period defined by the hardware specifications or when the CPU enters a specific state.

When the specified period has elapsed or the CPU exits the specified state, CPU operation automatically resumes and instruction execution begins from the next instruction.

The SNOOZE state is released under the following conditions:

- The predefined period of time passes
- A terminating exception occurs

Even if the conditions for acknowledging the above exceptions are not satisfied (due to the ID or NP value), as long as a SNOOZE mode release request exists, the SNOOZE state is released (for example, even if PSW.ID = 1, the SNOOZE state is released when INT0 occurs).

Note, however, that the SNOOZE mode will not be released if terminating exceptions are masked by the following mask settings, which are defined individually for each function:

- Terminating exceptions are masked by an interrupt channel mask setting specified by the interrupt controller[Note].
- Terminating exceptions are masked by a mask setting specified by using the floating-point operation exception enable bit.
- Terminating exceptions are masked by a mask setting defined by a hardware function other than the above.

Note This does not include masking specified by the ISPR and PMR registers.

[Description]

- The assembler generates one snooze machine instruction.

[Flag]

CY	---
OV	---
S	---
Z	---
SAT	---

5.9.9 Loop instructions

Next table lists the loop instructions.
See the RH850 product user's manual and architecture edition for details.

Table 5.43 Loop Instructions

Instruction	Meanings
loop	Loop

loop

Loop (Loop)

[Syntax]

- loop reg1, disp16

The following can be specified as disp16:

- Absolute expression having a value of up to 16 bits
- Relative expression having a PC offset reference of label
- label (PC offset)

[Function]

Updates the general-purpose register reg1 by adding -1 from its contents. If the contents after this update are not 0, the following processing is performed. If the contents are 0, the system continues to the next instruction.

- The result of logically shifting the 15-bit immediate data 1 bit to the left and zero-extending it to word length is subtracted from the current PC value, and then the control is transferred.
- -1 (0xFFFFFFFF) is added to general-purpose register reg1. The carry flag is updated in the same way as when the add instruction, not the sub instruction, is executed.

Remark "0" is implicitly used for bit 0 of the 16-bit displacement. Note that, because the current PC value used for calculation is the address of the first byte of this instruction, if the displacement value is 0, the branch destination is this instruction.

[Description]

- If any of the following is specified for disp16, the assembler generates one loop machine instruction.
  (a) Absolute expression having a value in the range of 0 to 65535
  (b) Relative expression that has a PC offset reference of a label having a definition in the same section of the same file as this instruction, and having a value in the range of 0 to 65535

[Flag]

CY	"1" if a carry occurs from MSB in the reg1 operation; otherwise, "0"
OV	"1" if an overflow occurs in the reg1 operation; otherwise, "0"
S	"1" if reg1 is negative; otherwise, "0"
Z	"1" if reg1 is 0; otherwise, "0"
SAT	---

[Caution]

- If an absolute expression having a value exceeding the range of 0 to 65535 or a relative expression having a PC offset reference of a label with a definition in the same section of the same file as this instruction and having a value exceeding the range of 0 to 65535 is specified as disp16, the assembler outputs the following message and stops assembling.

```
E0550230 : Illegal operand (range error in displacement).
```

- If an absolute expression having an odd-numbered value or a relative expression having a PC offset reference of a label with a definition in the same section of the same file as this instruction and having an odd-numbered value is specified as disp16, the assembler outputs the following message and stops assembling.

```
E0550226 : Illegal operand (must be even displacement).
```

- If r0 is specified as reg1 in the loop instruction, the assembler outputs the following message and stops assembling.

```
E0550240 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

- Do not specify r0 for reg1.

5.9.10 Floating-point operation instructions

Next table lists the floating-point operation instructions.
See the RH850 product user's manual and architecture edition for details.

Table 5.44 Floating-point Operation Instructions (Basic Operation Instructions)

Instruction	Meanings
absf.d	Floating-point absolute value (double precision)
absf.s	Floating-point absolute value (single precision)
addf.d	Floating-point add (double precision)
addf.s	Floating-point add (single precision)
divf.d	Floating-point division (double precision)
divf.s	Floating-point division (single precision)
maxf.d	Floating-point maximum value (double precision)
maxf.s	Floating-point maximum value (single precision)
minf.d	Floating-point minimum value (double precision)
minf.s	Floating-point minimum value (single precision)
mulf.d	Floating-point multiplication (double precision)
mulf.s	Floating-point multiplication (single precision)
negf.d	Floating-point sign inversion (double precision)
negf.s	Floating-point sign inversion (single precision)
recipf.d	Reciprocal (double precision)
recipf.s	Reciprocal (single precision)
rsqrtf.d	Reciprocal of square root (double precision)
rsqrtf.s	Reciprocal of square root (single precision)
sqrtf.d	Square root (double precision)
sqrtf.s	Square root (single precision)
subf.d	Floating-point subtraction (double precision)
subf.s	Floating-point subtraction (single precision)

Table 5.45 Floating-point Operation Instructions (Exchange Instructions)

Instruction	Meanings
ceilf.dl	Conversion to fixed-point format (double precision)
ceilf.dw	Conversion to fixed-point format (double precision)
ceilf.dul	Conversion to unsigned fixed-point format (double precision)
ceilf.duw	Conversion to unsigned fixed-point format (double precision)
ceilf.sl	Conversion to fixed-point format (single precision)
ceilf.sw	Conversion to fixed-point format (single precision)
ceilf.sul	Conversion to unsigned fixed-point format (single precision)

Instruction	Meanings
ceilf.suw	Conversion to unsigned fixed-point format (single precision)
cvtf.dl	Conversion to fixed-point format (double precision)
cvtf.ds	Conversion to floating-point format (double precision)
cvtf.dul	Conversion to unsigned fixed-point format (double precision)
cvtf.duw	Conversion to unsigned fixed-point format (double precision)
cvtf.dw	Conversion to fixed-point format (double precision)
cvtf.ld	Conversion to floating-point format (double precision)
cvtf.ls	Conversion to floating-point format (single precision)
cvtf.sd	Conversion to floating-point format (double precision)
cvtf.sl	Conversion to fixed-point format (single precision)
cvtf.sul	Conversion to unsigned fixed-point format (single precision)
cvtf.suw	Conversion to unsigned fixed-point format (single precision)
cvtf.sw	Conversion to fixed-point format (single precision)
cvtf.uld	Conversion to floating-point format (double precision)
cvtf.uls	Conversion to floating-point format (single precision)
cvtf.uwd	Conversion to floating-point format (double precision)
cvtf.uws	Conversion to floating-point format (single precision)
cvtf.wd	Conversion to floating-point format (double precision)
cvtf.ws	Conversion to floating-point format (single precision)
floorf.dl	Conversion to fixed-point format (double precision)
floorf.dw	Conversion to fixed-point format (double precision)
floorf.dul	Conversion to unsigned fixed-point format (double precision)
floorf.duw	Conversion to unsigned fixed-point format (double precision)
floorf.sl	Conversion to fixed-point format (single precision)
floorf.sw	Conversion to fixed-point format (single precision)
floorf.sul	Conversion to unsigned fixed-point format (single precision)
floorf.suw	Conversion to unsigned fixed-point format (single precision)
trncf.dl	Conversion to fixed-point format (double precision)
trncf.dul	Conversion to unsigned fixed-point format (double precision)
trncf.duw	Conversion to unsigned fixed-point format (double precision)
trncf.dw	Conversion to fixed-point format (double precision)
trncf.sl	Conversion to fixed-point format (single precision)
trncf.sul	Conversion to unsigned fixed-point format (single precision)
trncf.suw	Conversion to unsigned fixed-point format (single precision)
trncf.sw	Conversion to fixed-point format (single precision)

Table 5.46 Floating-point Operation Instructions (Compare Instructions)

Instruction	Meanings
cmpf.d	Floating-point compare (double)
cmpf.s	Floating-point compare (single)

Table 5.47 Floating-point Operation Instructions (Conditional Move Instructions)

Instruction	Meanings
cmovf.d	Conditional move (double precision)
cmovf.s	Conditional move (single precision)

Table 5.48 Floating-point Operation Instructions (Conditional Bit Move Instructions)

Instruction	Meanings
trfsr	Flag transfer

absf.d

Floating-point absolute value (double precision) (Floating-point Absolute Value (Double))

[Syntax]

- absf.d reg1, reg2

[Function]

This instruction takes the absolute value from the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores it in the register pair specified by general-purpose register reg2.

[Description]

- The assembler generates one absf.d machine instruction.

absf.s

Floating-point absolute value (single precision) (Floating-point Absolute Value (Single))

[Syntax]

- absf.s reg1, reg2

[Function]

This instruction takes the absolute value from the single-precision floating-point format contents of general-purpose register reg1, and stores it in general-purpose register reg2.

[Description]

- The assembler generates one absf.s machine instruction.

addf.d

Floating-point add (double precision) (Floating-point Add (Double))

[Syntax]

- addf.d reg1, reg2, reg3

[Function]

This instruction adds the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 with the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in the register pair specified by general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one addf.d machine instruction.

addf.s

Floating-point add (single precision) (Floating-point Add (Single))

[Syntax]

- addf.s reg1, reg2, reg3

[Function]

This instruction adds the single-precision floating-point format contents of general-purpose register reg1 with the single-precision floating-point format contents of general-purpose register reg2, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one addf.s machine instruction.

divf.d

Floating-point division (double precision) (Floating-point Divide (Double))

[Syntax]

- divf.d reg1, reg2, reg3

[Function]

This instruction divides double-precision floating-point format contents of the register pair specified by general-purpose register reg2 by the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores the result in the register pair specified by general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one divf.d machine instruction.

divf.s

Floating-point division (single precision) (Floating-point Divide (Single))

[Syntax]

- divf.s reg1, reg2, reg3

[Function]

This instruction divides the single-precision floating-point format contents of general-purpose register reg2 by the single-precision floating-point format contents of general-purpose register reg1, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one divf.s machine instruction.

maxf.d

Floating-point maximum value (double precision) (Floating-point Maximum (Double))

[Syntax]

- maxf.d reg1, reg2, reg3

[Function]

This instruction extracts the maximum value from the double-precision floating-point format data in the register pair specified by general-purpose registers reg1 and reg2, and stores it in the register pair specified by general-purpose register reg3.

If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, Q-NaN is stored and no exception occurs.

[Description]

- The assembler generates one maxf.d machine instruction.

maxf.s

Floating-point maximum value (single precision) (Floating-point Maximum (Single))

[Syntax]

- maxf.s reg1, reg2, reg3

[Function]

This instruction extracts the maximum value from the single-precision floating-point format data in general-purpose registers reg1 and reg2, and stores it in general-purpose register reg3.
If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected.
If invalid operation exceptions are not enabled, Q-NaN is stored and no exception occurs.

[Description]

- The assembler generates one maxf.s machine instruction.

minf.d

Floating-point minimum value (double precision) (Floating-point Minimum (Double))

[Syntax]

- minf.d reg1, reg2, reg3

[Function]

This instruction extracts the minimum value from the double-precision floating-point format data in the register pair specified by general-purpose registers reg1 and reg2, and stores it in the register pair specified by general-purpose register reg3.

If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, Q-NaN is stored and no exception occurs.

[Description]

- The assembler generates one minf.d machine instruction.

minf.s

Floating-point minimum value (single precision) (Floating-point Minimum (Single))

[Syntax]

- minf.s reg1, reg2, reg3

[Function]

This instruction extracts the minimum value from the single-precision floating-point format data in general-purpose registers reg1 and reg2, and stores it in general-purpose register reg3.
If one of the source operands is S-NaN, an IEEE754-defined invalid operation exception is detected.
If invalid operation exceptions are not enabled, Q-NaN is stored and no exception occurs.

[Description]

- The assembler generates one minf.s machine instruction.

mulf.d

Floating-point multiplication (double precision) (Floating-point Multiply (Double))

[Syntax]

- mulf.d reg1, reg2, reg3

[Function]

This instruction multiplies double-precision floating-point format contents of the register pair specified by general-purpose register reg2 by the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores the result in general-purpose register reg3.

[Description]

- The assembler generates one mulf.d machine instruction.

mulf.s

Floating-point multiplication (single precision) (Floating-point Multiply (Single))

[Syntax]

- mulf.s reg1, reg2, reg3

[Function]

This instruction multiplies the single-precision floating-point format contents of general-purpose register reg2 by the single-precision floating-point format contents of general-purpose register reg1, and stores the result in general-purpose register reg3.

[Description]

- The assembler generates one mulf.s machine instruction.

negf.d

Floating-point sign inversion (double precision) (Floating-point Negate (Double))

[Syntax]

- negf.d reg1, reg2

[Function]

This instruction inverts the sign of double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores the result in general-purpose register reg2.

[Description]

- The assembler generates one negf.d machine instruction.

negf.s

Floating-point sign inversion (single precision) (Floating-point Negate (Single))

[Syntax]

- negf.s reg1, reg2

[Function]

This instruction inverts the sign of the single-precision floating-point format contents of general-purpose register reg1, and stores the result in general-purpose register reg2.

[Description]

- The assembler generates one negf.s machine instruction.

recipf.d

Reciprocal (double precision) (Reciprocal of a Floating-point Value (Double))

[Syntax]

- recipf.d reg1, reg2

[Function]

This instruction approximates the reciprocal of the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores the result in the register pair specified by general-purpose register reg2. The result differs from the result obtained by using the divf instruction.

[Description]

- The assembler generates one recipf.d machine instruction.

recipf.s

Reciprocal (single precision) (Reciprocal of a Floating-point Value (Single))

[Syntax]

- recipf.s reg1, reg2

[Function]

This instruction approximates the reciprocal of the single-precision floating-point format contents of general-purpose register reg1, and stores the result in general-purpose register reg2. The result differs from the result obtained by using the divf instruction.

[Description]

- The assembler generates one recipf.s machine instruction.

rsqrtf.d

Reciprocal of square root (double precision) (Reciprocal of the Square Root of a Floating-point Value (Double))

[Syntax]

- rsqrtf.d reg1, reg2

[Function]

This instruction obtains the arithmetic positive square root of the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, then approximates the reciprocal of this result and stores the result in the register pair specified by general-purpose register reg2.

The result differs from the result obtained when using a combination of the sqrtf and divf instructions.

[Description]

- The assembler generates one rsqrtf.d machine instruction.

rsqrtf.s

Reciprocal of square root (single precision) (Reciprocal of the Square Root of a Floating-point Value (Single))

[Syntax]

- rsqrtf.s reg1, reg2

[Function]

This instruction obtains the arithmetic positive square root of the single-precision floating-point format contents of general-purpose register reg1, then approximates the reciprocal of this result and stores it in general-purpose register reg2.
The result differs from the result obtained when using a combination of the sqrtf and divf instructions.

[Description]

- The assembler generates one rsqrtf.s machine instruction.

sqrtf.d

Square root (double precision) (Floating-point Square Root (Double))

[Syntax]

- sqrtf.d reg1, reg2

[Function]

This instruction obtains the arithmetic positive square root of the double-precision floating-point format contents of the register pair specified by general-purpose register reg1, and stores the result in the register pair specified by general-purpose register reg2. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode. When the source operand value is -0, the result becomes -0.

[Description]

- The assembler generates one sqrtf.d machine instruction.

sqrtf.s

Square root (single precision) (Floating-point Square Root (Single))

[Syntax]

- sqrtf.s reg1, reg2

[Function]

This instruction obtains the arithmetic positive square root of the single-precision floating-point format contents of general-purpose register reg1, and stores it in general-purpose register reg2. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode. When the source operand value is -0, the result becomes -0.

[Description]

- The assembler generates one sqrtf.s machine instruction.

subf.d

Floating-point subtraction (double precision) (Floating-point Subtract (Double))

[Syntax]

- subf.d reg1, reg2, reg3

[Function]

This instruction subtracts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 from the double-precision floating-point format contents of the register pair specified by general-purpose register reg2, and stores the result in the register pair specified by general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one subf.d machine instruction.

subf.s

Floating-point subtraction (single precision) (Floating-point Subtract (Single))

[Syntax]

- subf.s reg1, reg2, reg3

[Function]

This instruction subtracts the single-precision floating-point format contents of general-purpose register reg1 from the single-precision floating-point format contents of general-purpose register reg2, and stores the result in general-purpose register reg3. The operation is executed as if it were of infinite accuracy, and the result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one subf.s machine instruction.

ceilf.dl

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Long, round toward positive (Double))

[Syntax]

- ceilf.dl reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one ceilf.dl machine instruction.

ceilf.dw

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Word, round toward positive (Double))

[Syntax]

- ceilf.dw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one ceilf.dw machine instruction.

ceilf.dul

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Long, round toward positive (Double))

[Syntax]

- ceilf.dul reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0,

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one ceilf.dul machine instruction.

ceilf.duw

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Word, round toward positive (Double))

[Syntax]

- ceilf.duw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one ceilf.duw machine instruction.

ceilf.sl

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Long, round toward positive (Single))

[Syntax]

- ceilf.sl reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one ceilf.sl machine instruction.

ceilf.sw

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Word, round toward positive (Single))

[Syntax]

- ceilf.sw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one ceilf.sw machine instruction.

ceilf.sul

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Long, round toward positive (Single))

[Syntax]

- ceilf.sul reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents specified by general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one ceilf.sul machine instruction.

ceilf.suw

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Word, round toward positive (Single))

[Syntax]

- ceilf.suw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the +∞ direction regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one ceilf.suw machine instruction.

cvtf.dl

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Long (Double))

[Syntax]

- cvtf.dl reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 64-bit fixed-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one cvtf.dl machine instruction.

cvtf.ds

Conversion to floating-point format (double precision) (Floating-point Convert Double to Single (Double))

[Syntax]

- cvtf.ds reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to single-precision floating-point format, and stores the result in general-purpose register reg2. The result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one cvtf.ds machine instruction.

cvtf.dul

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Long (Double))

[Syntax]

- cvtf.dul reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 64-bit fixed-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one cvtf.dul machine instruction.

cvtf.duw

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Word (Double))

[Syntax]

- cvtf.duw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one cvtf.duw machine instruction.

cvtf.dw

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Word (Double))

[Syntax]

- cvtf.dw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one cvtf.dw machine instruction.

cvtf.ld

Conversion to floating-point format (double precision) (Floating-point Convert Long to Double (Double))

[Syntax]

- cvtf.ld reg1, reg2

[Function]

This instruction arithmetically converts the 64-bit fixed-point format contents of the register pair specified by general-purpose register reg1 to double-precision floating-point format in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

[Description]

- The assembler generates one cvtf.ld machine instruction.

cvtf.ls

Conversion to floating-point format (single precision) (Floating-point Convert Long to Single (Single))

[Syntax]

- cvtf.ls reg1, reg2

[Function]

This instruction arithmetically converts the 64-bit fixed-point format contents of the register pair specified by general-purpose register reg1 to single-precision floating-point format, and stores the result in general-purpose register reg2. The result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one cvtf.ls machine instruction.

cvtf.sd

Conversion to floating-point format (double precision) (Floating-point Convert Single to Double (Double))

[Syntax]

- cvtf.sd reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to double-precision floating-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

[Description]

- The assembler generates one cvtf.sd machine instruction.

cvtf.sl

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Long (Single))

[Syntax]

- cvtf.sl reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 64-bit fixed-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one cvtf.sl machine instruction.

cvtf.sul

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Long (Single))

[Syntax]

- cvtf.sul reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to unsigned 64-bit fixed-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one cvtf.sul machine instruction.

cvtf.suw

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Word (Single))

[Syntax]

- cvtf.suw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one cvtf.suw machine instruction.

cvtf.sw

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Word (Single))

[Syntax]

- cvtf.sw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one cvtf.sw machine instruction.

cvtf.uld

Conversion to floating-point format (double precision) (Floating-point Convert Unsigned-Long to Double (Double))

[Syntax]

- cvtf.uld reg1, reg2

[Function]

This instruction arithmetically converts the unsigned 64-bit fixed-point format contents of the register pair specified by general-purpose register reg1 to double-precision floating-point format in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

[Description]

- The assembler generates one cvtf.uld machine instruction.

cvtf.uls

Conversion to floating-point format (single precision) (Floating-point Convert Unsigned-Long to Single (Single))

[Syntax]

- cvtf.uls reg1, reg2

[Function]

This instruction arithmetically converts the unsigned 64-bit fixed-point format contents of the register pair specified by general-purpose register reg1 to single-precision floating-point format, and stores the result in general-purpose register reg2. The result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one cvtf.uls machine instruction.

cvtf.uwd

Conversion to floating-point format (double precision) (Floating-point Convert Unsigned-Word to Double (Double))

[Syntax]

- cvtf.uwd reg1, reg2

[Function]

This instruction arithmetically converts the unsigned 32-bit fixed-point format contents of general-purpose register reg1 to double-precision floating-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

This conversion operation is performed accurately, without any loss of precision.

[Description]

- The assembler generates one cvtf.uwd machine instruction.

cvtf.uws

Conversion to floating-point format (single precision) (Floating-point Convert Unsigned-Word to Single (Single))

[Syntax]

- cvtf.uws reg1, reg2

[Function]

This instruction arithmetically converts the unsigned 32-bit fixed-point format contents of general-purpose register reg1 to single-precision floating-point format, and stores the result in general-purpose register reg2. The result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one cvtf.uws machine instruction.

cvtf.wd

Conversion to floating-point format (double precision) (Floating-point Convert Word to Double (Double))

[Syntax]

- cvtf.wd reg1, reg2

[Function]

This instruction arithmetically converts the 32-bit fixed-point format contents of general-purpose register reg1 to double-precision floating-point format, in accordance with the current rounding mode, and stores the result in the register pair specified by general-purpose register reg2.

This conversion operation is performed accurately, without any loss of precision.

[Description]

- The assembler generates one cvtf.wd machine instruction.

cvtf.ws

Conversion to floating-point format (single precision) (Floating-point Convert Word to Single (single))

[Syntax]

- cvtf.ws reg1, reg2

[Function]

This instruction arithmetically converts the 32-bit fixed-point format contents of general-purpose register reg1 to single-precision floating-point format, and stores the result in general-purpose register reg2. The result is rounded in accordance with the current rounding mode.

[Description]

- The assembler generates one cvtf.ws machine instruction.

floorf.dl

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Long, round toward negative (Double))

[Syntax]

- floorf.dl reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one floorf.dl machine instruction.

floorf.dw

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Word, round toward negative (Double))

[Syntax]

- floorf.dw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one floorf.dw machine instruction.

floorf.dul

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Long, round toward negative (Double))

[Syntax]

- floorf.dul reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one floorf.dul machine instruction.

floorf.duw

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Word, round toward negative (Double))

[Syntax]

- floorf.duw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one floorf.duw machine instruction.

floorf.sl

Conversion to fixed-point format (single precision) (Floating-point Convert Double to Long, round toward negative (Single))

[Syntax]

- floorf.sl reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one floorf.sl machine instruction.

floorf.sw

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Word, round toward negative (Single))

[Syntax]

- floorf.sw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one floorf.sw machine instruction.

floorf.sul

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Long, round toward negative (Single))

[Syntax]

- floorf.sul reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one floorf.sul machine instruction.

floorf.suw

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Word, round toward negative (Single))

[Syntax]

- floorf.suw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the -∞ direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one floorf.suw machine instruction.

trncf.dl

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Long, round toward zero (Double))

[Syntax]

- trncf.dl reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{63} - 1 to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{63} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one trncf.dl machine instruction.

trncf.dul

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Long, round toward zero (Double))

[Syntax]

- trncf.dul reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one trncf.dul machine instruction.

trncf.duw

Conversion to unsigned fixed-point format (double precision) (Floating-point Convert Double to Unsigned-Word, round toward zero (Double))

[Syntax]

- trncf.duw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one trncf.duw machine instruction.

trncf.dw

Conversion to fixed-point format (double precision) (Floating-point Convert Double to Word, round toward zero (Double))

[Syntax]

- trncf.dw reg1, reg2

[Function]

This instruction arithmetically converts the double-precision floating-point format contents of the register pair specified by general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one trncf.dw machine instruction.

trncf.sl

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Long, round toward zero (Single))

[Syntax]

- trncf.sl reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of $2^{63} - 1$ to -2^{63}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : $2^{63} - 1$ is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{63} is returned.

[Description]

- The assembler generates one trncf.sl machine instruction.

trncf.sul

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Long, round toward zero (Single))

[Syntax]

- trncf.sul reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point format contents of general-purpose register reg1 to unsigned 64-bit fixed-point format, and stores the result in the register pair specified by general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{64} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{64} - 1 to 0, or +∞ : 2^{64} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one trncf.sul machine instruction.

trncf.suw

Conversion to unsigned fixed-point format (single precision) (Floating-point Convert Single to Unsigned-Word, round toward zero (Single))

[Syntax]

- trncf.suw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point number format contents of general-purpose register reg1 to unsigned 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite, not-a-number, or negative number, or when the rounded result is outside the range of 2^{32} - 1 to 0, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number outside the range of 2^{32} - 1 to 0, or +∞ : 2^{32} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : 0 is returned.

[Description]

- The assembler generates one trncf.suw machine instruction.

trncf.sw

Conversion to fixed-point format (single precision) (Floating-point Convert Single to Word, round toward zero (Single))

[Syntax]

- trncf.sw reg1, reg2

[Function]

This instruction arithmetically converts the single-precision floating-point number format contents of general-purpose register reg1 to 32-bit fixed-point format, and stores the result in general-purpose register reg2.

The result is rounded in the zero direction, regardless of the current rounding mode.

When the source operand is infinite or not-a-number, or when the rounded result is outside the range of 2^{31} - 1 to -2^{31}, an IEEE754-defined invalid operation exception is detected.

If invalid operation exceptions are not enabled, the preservation bit (bit 4) of the FPSR register is set as an invalid operation and no exception occurs. The return value differs as follows, according to differences among sources.

- Source is a positive number or +∞ : 2^{31} - 1 is returned.
- Source is a negative number, not-a-number, or -∞ : -2^{31} is returned.

[Description]

- The assembler generates one trncf.sw machine instruction.

cmpf.d

Floating-point compare (double)

[Syntax]

- cmpf.d imm4, reg1, reg2, cc#3
- cmpf*cnd*.d reg1, reg2

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits

[Function]

- Syntax "cmpf.d imm4, reg1, reg2, cc#3"
  The content in double-precision floating-point format in the register pair specified by reg2 is compared with the content in double-precision floating-point format in the register pair specified by reg1, via the imm4 comparison condition. The result (1 if true; 0 if false) is set in the condition bit (CC(7:0) bits; bits 31-24) in the FPSR register specified via cc#3. If cc#3 is omitted, it is set in the CC0 bit (bit 24).
- Syntax "cmpf*cnd*.d reg1, reg2"
  Via cmpf*cnd*.d, a corresponding "cmpf.d" instruction is generated (see "Table 5.49 cmpfcnd.d Instruction List" for details), and expanded in the format "cmpf.d imm4, reg1, reg2, cc#3". The content in single-precision floating-point format in the register pair specified by reg2 is compared with the content in single-precision floating-point format in the register pair specified by reg1, via the comparison condition. The result (1 if true; 0 if false) is set in the condition bit (CC(7:0) bits; bits 31-24) in the FPSR register specified via cc#3. If cc#3 is omitted, it is set in the CC0 bit (bit 24).

[Description]

- If the instruction is executed in syntax "cmpf.d imm4, reg1, reg2, cc#3", the assembler generates one cmpf.d machine instruction.
- If the instruction is executed in syntax "cmpf*cnd*.d reg1, reg2", the assembler generates the corresponding cmpf.d instruction (see "Table 5.49 cmpfcnd.d Instruction List") and expands it to syntax "cmpf.d imm4, reg1, reg2, cc#3".

Table 5.49 cmpf*cnd*.d Instruction List

Instruction	Condition	Meaning of Condition	Instruction Expansion
cmpff.d	FALSE	Always false	cmpf.d 0x0
cmpfun.d	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.d 0x1
cmpfeq.d	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.d 0x2
cmpfueq.d	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.d 0x3
cmpfolt.d	reg2 < reg1	Neither is a non-number, and less than	cmpf.d 0x4
cmpfult.d	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.d 0x5
cmpfole.d	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.d 0x6
cmpfule.d	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.d 0x7
cmpfsf.d	FALSE	Always false	cmpf.d 0x8
cmpfngle.d	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.d 0x9
cmpfseq.d	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.d 0xA
cmpfngl.d	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.d 0xB

Instruction	Condition	Meaning of Condition	Instruction Expansion
cmpflt.d	reg2 < reg1	Neither is a non-number, and less than	cmpf.d 0xC
cmpfnge.d	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.d 0xD
cmpfle.d	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.d 0xE
cmpfngt.d	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.d 0xF

Remark ?: Unordered

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the cmpf.d instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

cmpf.s

Floating-point compare (single)

[Syntax]

- cmpf.s imm4, reg1, reg2, cc#3
- cmpf*cnd*.s reg1, reg2

The following can be specified for imm4:

- Absolute expression having a value up to 4 bits

[Function]

- Syntax "cmpf.s imm4, reg1, reg2, cc#3"
  The content in single-precision floating-point format in the register pair specified by reg2 is compared with the content in single-precision floating-point format in the register pair specified by reg1, via the imm4 comparison condition. The result (1 if true; 0 if false) is set in the condition bit (CC(7:0) bits; bits 31-24) in the FPSR register specified via cc#3. If cc#3 is omitted, it is set in the CC0 bit (bit 24).
- Syntax "cmpf*cnd*.s reg1, reg2"
  Via cmpf*cnd*.s, a corresponding "cmpf.s" instruction is generated (see "Table 5.50 cmpfcnd.s Instruction List" for details), and expanded in the format "cmpf.s imm4, reg1, reg2, cc#3". The content in single-precision floating-point format in the register pair specified by reg2 is compared with the content in single-precision floating-point format in the register pair specified by reg1, via the comparison condition. The result (1 if true; 0 if false) is set in the condition bit (CC(7:0) bits; bits 31-24) in the FPSR register specified via cc#3. If cc#3 is omitted, it is set in the CC0 bit (bit 24).

[Description]

- If the instruction is executed in syntax "cmpf.s imm4, reg1, reg2, cc#3", the assembler generates one cmpf.s machine instruction.
- If the instruction is executed in syntax "cmpf*cnd*.s reg1, reg2", the assembler generates the corresponding cmpf.s instruction (see "Table 5.50 cmpfcnd.s Instruction List") and expands it to syntax "cmpf.s imm4, reg1, reg2, cc#3".

Table 5.50 cmpf*cnd*.s Instruction List

Instruction	Condition	Meaning of Condition	Instruction Expansion
cmpff.s	FALSE	Always false	cmpf.s 0x0
cmpfun.s	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.s 0x1
cmpfeq.s	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.s 0x2
cmpfueq.s	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.s 0x3
cmpfolt.s	reg2 < reg1	Neither is a non-number, and less than	cmpf.s 0x4
cmpfult.s	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.s 0x5
cmpfole.s	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.s 0x6
cmpfule.s	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.s 0x7
cmpfsf.s	FALSE	Always false	cmpf.s 0x8
cmpfngle.s	Unordered	At least one of reg1 and reg2 is a non-number	cmpf.s 0x9
cmpfseq.s	reg2 = reg1	Neither is a non-number, and they are equal	cmpf.s 0xA
cmpfngl.s	reg2 ?= reg1	At least one is a non-number, or they are equal	cmpf.s 0xB

Instruction	Condition	Meaning of Condition	Instruction Expansion
cmpflt.s	reg2 < reg1	Neither is a non-number, and less than	cmpf.s 0xC
cmpfnge.s	reg2 ?< reg1	At least one is a non-number, or less than	cmpf.s 0xD
cmpfle.s	reg2 <= reg1	Neither is a non-number, and less than or equal	cmpf.s 0xE
cmpfngt.s	reg2 ?<= reg1	At least one is a non-number, or less than or equal	cmpf.s 0xF

Remark ?: Unordered

[Caution]

- If an absolute expression having a value exceeding 4 bits is specified as imm4 of the cmpf.s instruction, the following message is output, and assembly continues using the lower 4 bits of the specified value.

```
W0550011 : illegal operand (range error in immediate).
```

cmovf.d

Conditional move (double precision) (Floating-point Conditional Move (Double))

[Syntax]

- cmovf.d cc#3, reg1, reg2, reg3

[Function]

When the CC(7:0) bits of the FPSR register specified by fcbit in the opcode are true (1), data from the register pair specified by reg1 is stored in the register pair specified by reg3. When these bits are false (0), data from the register pair specified by reg2 is stored in the register pair specified by reg3.

[Description]

- The assembler generates one cmovf.d machine instruction.

[Caution]

- If r0 is specified as reg3 in the cmovf.d instruction, the assembler outputs the following message and stops assembling.

```
E0550262 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

cmovf.s

Conditional move (single precision) (Floating-point Conditional Move (Single))

[Syntax]

- cmovf.s cc#3, reg1, reg2, reg3

[Function]

When the CC(7:0) bits of the FPSR register specified by fcbit in the opcode are true (1), data from reg1 is stored in reg3. When these bits are false (0), the reg2 data is stored in reg3.

[Description]

- The assembler generates one cmovf.s machine instruction.

[Caution]

- If r0 is specified as reg3 in the cmovf.s instruction, the assembler outputs the following message and stops assembling.

```
E0550262 : Illegal operand (cannot use r0 as destination in RH850 mode).
```

trfsr

Flag transfer (Transfers specified CC bit to Zero flag in PSW (Single))

[Syntax]

- trfsr cc#3
- trfsr

[Function]

This instruction transfers the condition bits (the CC(7:0) bits: bits 31 to 24) in the FPSR register specified by fcbit to the Z flag in the PSW. If fcbit is omitted, this instruction transfers the CC0 bit (bit 24).

[Description]

- The assembler generates one trfsr machine instruction.

5.9.11 Other instructions

Other instructions are shown below.

- Virtualization support feature instructions
- Hardware multithreading feature instructions
- Cache instructions
- MMU control instructions
- SIMD instructions

Next table lists the floating-point operation instructions.
See the RH850 product user's manual and architecture edition for details.

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.

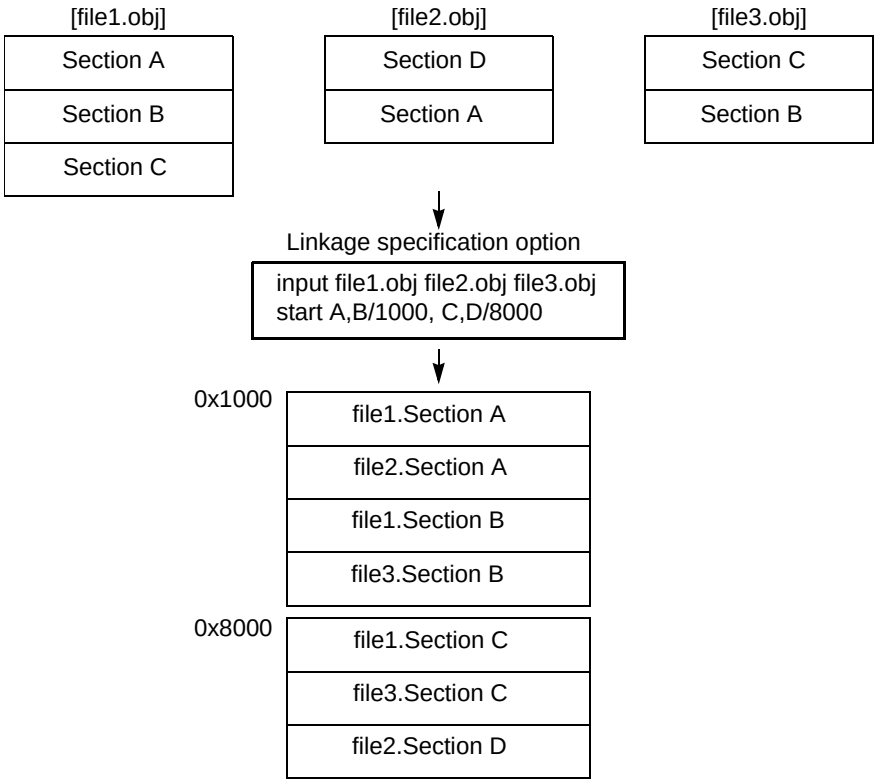
Even if the #pragma directive is not used to specify a section, however, allocation by the compiler to a particular section may already be set as the default setting in the program code or data (variables).

6.1.1 Section concatenation

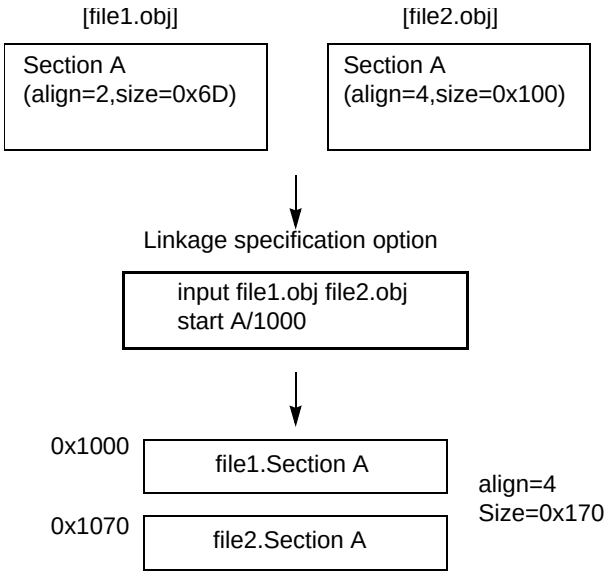
The optimizing linker (hereafter abbreviated "rlink") concatenates identical sections in the input relocatable files, and allocates them to the address specified by the -start option.

(1) Section allocation via the -start option

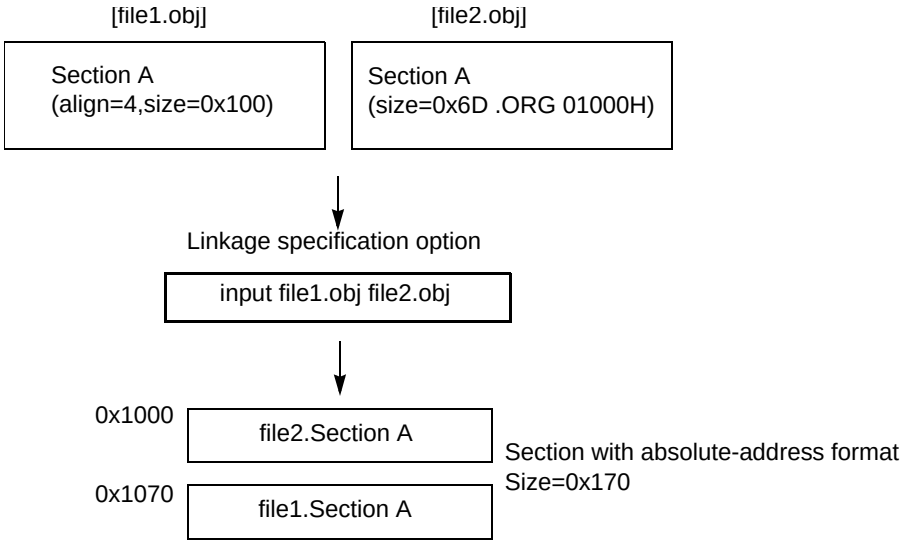
(a) Sections in different files with the same name are concatenated and allocated in the order of file input.



(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.

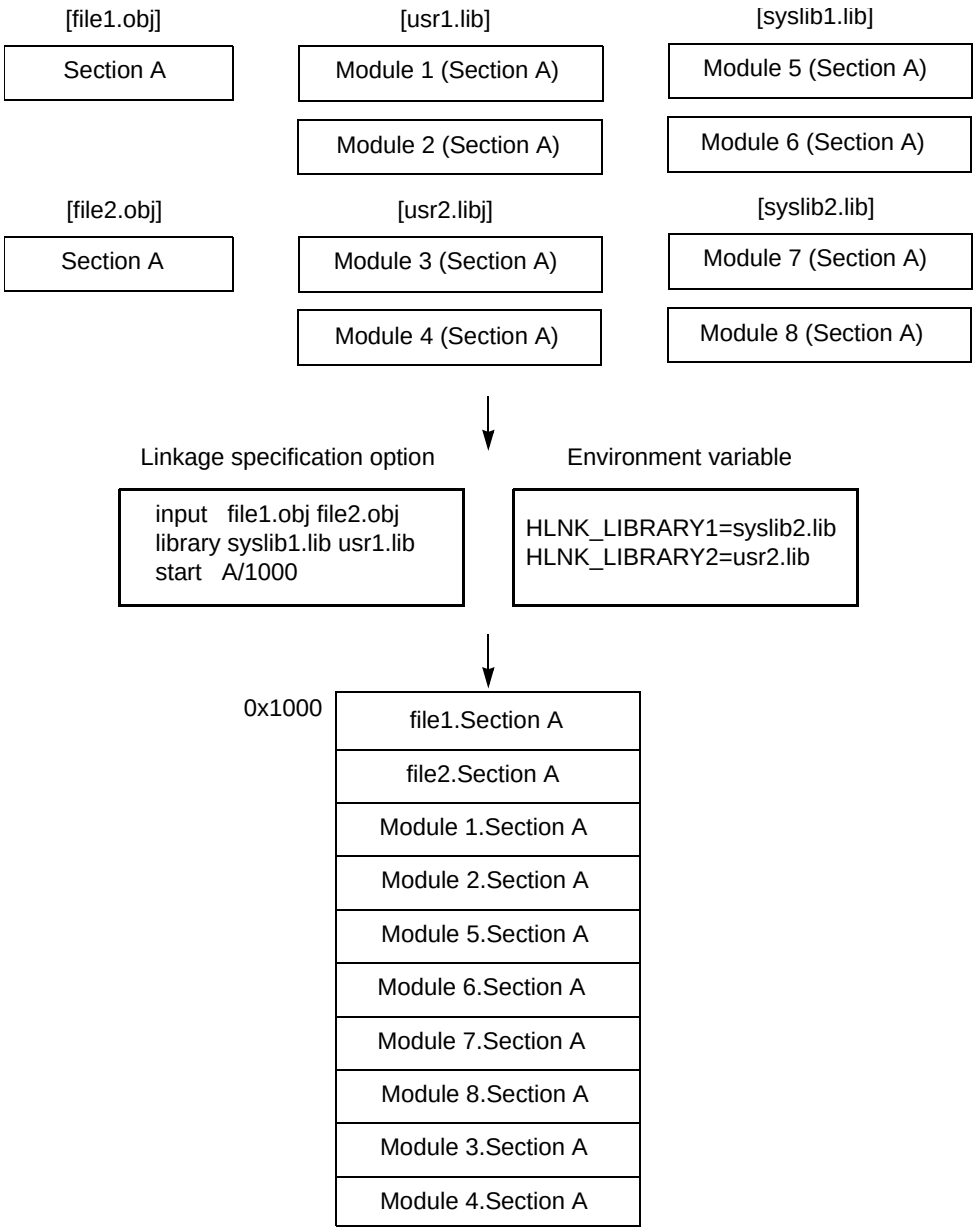


(c) If sections with the same name include both absolute-address format and relative-address format, then the sections with relative-address format are concatenated after the sections with absolute-address format.



(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 name of the reserved symbol with the value of the start address of a section is the name of that output section, preceded by the string "__s".

The name of the reserved symbol with the value of the first address after the end of a section is the name of that output section, preceded by the string "__e".

Below are shown the reserved sections and the special symbols corresponding to those sections.

Table 6.1 Special Symbol for Reserved Section

Reserved Section	Special Symbol for Reserved Section
.text	__s.text, __e.text
.bss	__s.bss, __e.bss
.zbss	__s.zbss, __e.zbss

Reserved Section	Special Symbol for Reserved Section
.zbss23	__s.zbss23, __e.zbss23
.ebss	__s.ebss, __e.ebss
.ebss23	__s.ebss23, __e.ebss23
.tbss4	__s.tbss4, __e.tbss4
.tbss5	__s.tbss5, __e.tbss5
.tbss7	__s.tbss7, __e.tbss7
.tbss8	__s.tbss8, __e.tbss8
.sbss	__s.sbss, __e.sbss
.sbss23	__s.sbss23, __e.sbss23
.data	__s.data, __e.data
.zdata	__s.zdata, __e.zdata
.zdata23	__s.zdata23, __e.zdata23
.edata	__s.edata, __e.edata
.edata23	__s.edata23, __e.edata23
.tdata	__s.tdata, __e.tdata
.tdata4	__s.tdata4, __e.tdata4
.tdata5	__s.tdata5, __e.tdata5
.tdata7	__s.tdata7, __e.tdata7
.tdata8	__s.tdata8, __e.tdata8
.sdata	__s.sdata, __e.sdata
.sdata23	__s.sdata23, __e.sdata23
.const	__s.const, __e.const
.zconst	__s.zconst, __e.zconst
.zconst23	__s.zconst23, __e.zconst23

Caution Only the symbols in the table for which there is a section in the post-linking executable file are generated.

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

Supplied Libraries	Library Name	Outline
Standard library	libc.lib	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
	libsetjmp.lib	Non-local jump functions
Mathematical library (doble-precision)	libm.lib	Mathematical functions
Mathematical library (sngle-precision)	libmf.lib	
Standard, double-precision, and single-precision mathematical library	rhf8n.lib rhs8n.lib rhf4n.lib rhs4n.lib rhf8z.lib rhf4z.lib	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 Mathematical functions

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 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.

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 7.2 Header Files

File Name	Outline
assert.h	Header file for program diagnostics
ctype.h	Header file for character conversion and classification
errno.h	Header file for reporting error condition
float.h	Header file for floating-point representation and floating-point operation
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 ANSI 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
_h_c_lib.h	Header file for the initial setting routine

7.3 Re-entrant

"Re-entrant" means that the function can "re-enter". A re-entrant function can be correctly executed even if an attempt is made in another process to execute that function 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. A function that must use RAM as a temporary area may not necessarily be re-entrant.

7.4 Library Function

This section explains Library Function.

7.4.1 Program diagnostic functions

Program diagnostic functions are as follows

Table 7.3 Program Diagnostic Function

Function/Macro Name	Outline
assert	Adds diagnostic features to the program

assert

Adds diagnostic features to the program.

[Classification]

Standard library

[Syntax]

#include <asssert.h>
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 7.4 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 7.5 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. strtok saves the pointer indicating the subsequent character. If the null pointer is used as the value of the first argument, a code that is not re-entrant is returned. This can be avoided by preserving the address of the last delimiting character in the application program, and passing *s* as an argument that is not vacant, by using this address.

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 according to the correspondence relationship of the processing system definition. The value of *errnum* is usually the duplication of global variable errno. Do not change the specified array of the application program.

7.4.4 Memory management functions

Memory management functions are as follows.

Table 7.6 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 7.7 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 7.8 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 7.9 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 in an implementation-defined format. 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. This is an implementation definition. 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 7.10 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
calloc	Memory allocation (initialized to zero)
malloc	Memory allocation(not initialized to zero)
realloc	Memory re-allocation
free	Memory release
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]

The operation is undefined when the -Xansi option is 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]

The operation is undefined when the -Xansi option is 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]

The operation is undefined when the -Xansi option is 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 re-entrant

[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]

The operation is undefined when the -Xansi option is 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]

The operation is undefined when the -Xansi option is 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 re-entrant.

[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 re-entrant.

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);
}
```

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".

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) {
        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 7.11 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.

7.4.10 Mathematical functions

Mathematical functions are as follows.

Table 7.12 Mathematical Functions

Function/Macro Name	Outline
expf	Exponent function
exp	Exponent function
logf	Logarithmic function (natural logarithm)
log	Logarithmic function (natural logarithm)
log10f	Logarithmic function (base = 10)
log10	Logarithmic function (base = 10)
powf	Power function
pow	Power function
sqrtf	Square root function
sqrt	Square root function
ceilf	ceiling function
ceil	ceiling function
fabsf	Absolute value function
fabs	Absolute value function
floorf	floor function
floor	floor function
fmodf	Remainder function
fmod	Remainder function
frexpf	Divide floating-point number into mantissa and power
frexp	Divide floating-point number into mantissa and power
ldexpf	Convert floating-point number to power
ldexp	Convert floating-point number to power
modff	Divide floating-point number into integer and decimal
modf	Divide floating-point number into integer and decimal
cosf	Cosine
cos	Cosine
sinf	Sine
sin	Sine
tanf	Tangent
tan	Tangent
acosf	Arc cosine
acos	Arc cosine

Function/Macro Name	Outline
asinf	Arc sine
asin	Arc sine
atanf	Arc tangent
atan	Arc tangent
atan2f	Arc tangent (y / x)
atan2	Arc tangent (y / x)
coshf	Hyperbolic cosine
cosh	Hyperbolic cosine
sinhf	Hyperbolic sine
sinh	Hyperbolic sine
tanhf	Hyperbolic tangent
tanh	Hyperbolic tangent

expf

Exponent function

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   expf(float x);
```

[Return value]

Returns the *x*th power of e.

expf returns an denormal number if an underflow occurs (if *x* is a negative number that cannot express the result), and sets macro ERANGE to global variable errno. If an overflow occurs (if *x* is too great a number), HUGE_VAL (maximum double type numerics that can be expressed) is returned, and macro ERANGE is set to global variable errno.

[Description]

This function calculates the *x*th power of e (e is the base of a natural logarithm and is about 2.71828).

exp

Exponent function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  exp(double x);
```

[Return value]

Returns the *x*th power of e.

expf returns an denormal number if an underflow occurs (if *x* is a negative number that cannot express the result), and sets macro ERANGE to global variable errno. If an overflow occurs (if *x* is too great a number), HUGE_VAL (maximum double type numerics that can be expressed) is returned, and macro ERANGE is set to global variable errno.

[Description]

This function calculates the *x*th power of e (e is the base of a natural logarithm and is about 2.71828).

logf

Logarithmic function (natural logarithm)

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   logf(float x);
```

[Return value]

Returns the natural logarithm of *x*.

logf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is negative. If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of *x*, i.e., logarithm with base e.

log

Logarithmic function (natural logarithm)

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  log(double x);
```

[Return value]

Returns the natural logarithm of *x*.

logf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is negative. If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the natural logarithm of *x*, i.e., logarithm with base e.

log10f

Logarithmic function (base = 10)

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   log10f(float x);
```

[Return value]

Returns the logarithm of *x* with base 10.
log10f returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is negative. If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the logarithm of *x* with base 10. This is realized by "log (*x*) / log (10)".

log10

Logarithmic function (base = 10)

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double   log10(double x);
```

[Return value]

Returns the logarithm of *x* with base 10.
log10f returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is negative. If *x* is zero, it returns -∞ and sets macro ERANGE to global variable errno.

[Description]

This function calculates the logarithm of *x* with base 10. This is realized by "log (*x*) / log (10)".

powf

Power function

[Classification]

Mathematical library

[Syntax]

#include <mathf.h>
float powf(float *x*, float *y*);

[Return value]

Returns the *y*th power of *x*.
powf returns a negative solution only if *x* < 0 and *y* is an odd integer. If *x* < 0 and *y* is a non-integer or if *x* = *y* = 0, powf returns a Not a Nuber(NaN) and sets the macro EDOM for the global variable errno. If *x* = 0 and *y* < 0 or if an overflow occurs, powf returns ±HUGE_VAL and sets the macro ERANGE for errno. If the solution vanished approaching zero, powf returns 0 and sets the macro ERANGE for errno. If the solution is a denormal number, powf sets the macro ERANGE for errno.

[Description]

This function calculates the *y*th power of *x*.

[Example]

```
#include    <mathf.h>
float func(void) {
        float   ret, x, y;
        ret = powf(x, y);   /*Returns yth power of x to ret.*/
          :
        return(ret);
}
```

pow

Power function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  pow(double x, double y);
```

[Return value]

Returns the *y*th power of *x*.

powf returns a negative solution only if $x < 0$ and *y* is an odd integer. If $x < 0$ and *y* is a non-integer or if $x = y = 0$, powf returns a Not a Nuber(NaN) and sets the macro EDOM for the global variable errno. If $x = 0$ and $y < 0$ or if an overflow occurs, powf returns ±HUGE_VAL and sets the macro ERANGE for errno. If the solution vanished approaching zero, powf returns 0 and sets the macro ERANGE for errno. If the solution is a denormal number, powf sets the macro ERANGE for errno.

[Description]

This function calculates the *y*th power of *x*.

sqrtf

Square root function

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   sqrtf(float x);
```

[Return value]

Returns the positive square root of *x*.
sqrtf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is a negative real number.

[Description]

This function calculates the square root of *x*.

sqrt

Square root function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  sqrt(double x);
```

[Return value]

Returns the positive square root of *x*.
sqrtf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if *x* is a negative real number.

[Description]

This function calculates the square root of *x*.

ceilf

ceiling function

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   ceilf(float x);
```

[Return value]

Returns the minimum integer greater than *x* and *x*.

[Description]

This function calculates the minimum integer value greater than *x* and *x*.

ceil

ceiling function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  ceil(double x);
```

[Return value]

Returns the minimum integer greater than *x* and *x*.

[Description]

This function calculates the minimum integer value greater than *x* and *x*.

fabsf

Absolute value function

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   fabsf(float x);
```

[Return value]

Returns the absolute value (size) of *x*.

[Description]

This function calculates the absolute value (size) of *x* by directly manipulating the bit representation of *x*.

fabs

Absolute value function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  fabs(double x);
```

[Return value]

Returns the absolute value (size) of *x*.

[Description]

This function calculates the absolute value (size) of *x* by directly manipulating the bit representation of *x*.

floorf

floor function

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   floorf(float x);
```

[Return value]

Returns the maximum integer value less than *x* and *x*.

[Description]

This function calculates the maximum integer value less than *x* and *x*.

floor

floor function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  floor(double x);
```

[Return value]

Returns the maximum integer value less than *x* and *x*.

[Description]

This function calculates the maximum integer value less than *x* and *x*.

fmodf

Remainder function

[Classification]

Mathematical library

[Syntax]

#include <mathf.h>
float fmodf(float *x*, float *y*);

[Return value]

Returns a floating-point value that is the remainder resulting from dividing *x* by *y*.
If *y* is ±∞, fmodf returns *x*.
If *x* is ±∞ or *y* is zero, fmodf returns a Not a Nuber(NaN) and sets macro ERANGE to global variable errno.

[Description]

This function calculates a floating-point value that is the remainder resulting from dividing *x* by *y*. In other words, it calculates the value "*x* - i * *y*" for the maximum integer i that has a sign the same as *x* and is less than *y*, if *y* is not zero.

[Example]

```
#include    <mathf.h>
void func(void) {
        float   ret, x, y;
       ret = fmodf(x, y);  /*Returns remainder resulting from dividing x by y to ret.*/
          :
}
```

fmod

Remainder function

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  fmod(double x, double y);
```

[Return value]

Returns a floating-point value that is the remainder resulting from dividing *x* by *y*.
If *y* is ±∞, fmod returns *x*.
If *x* is ±∞ or *y* is zero, fmod returns a Not a Nuber(NaN) and sets macro ERANGE to global variable errno.

[Description]

This function calculates a floating-point value that is the remainder resulting from dividing *x* by *y*. In other words, it calculates the value "*x* - i * *y*" for the maximum integer i that has a sign the same as *x* and is less than *y*, if *y* is not zero.

frexpf

Divide floating-point number into mantissa and power

[Classification]

Mathematical library

[Syntax]

#include <mathf.h>
float frexpf(float *val*, int **exp*);

[Return value]

Returns mantissa m.
frexpf sets 0 to **exp* and returns 0 if *val* is 0.
If *val* is ±∞, frexpf returns zero and sets macro EDOM to global variable errno.

[Description]

This function expresses *val* of float type as mantissa m and the pth power of 2. The resulting mantissa m is 0.5 <= | x | < 1.0, unless *val* is zero. p is stored in **exp*. m and p are calculated so that *val* = m * 2^p.

[Example]

```
#include    <mathf.h>
void func(void) {
        float   ret, x;
        int     exp;
        x = 5.28;
        ret = frexpf(x, &exp);  /*Resultant mantissa 0.66 is returned to ret, and 3 is
                                  stored in exp.*/
          :
}
```

frexp

Divide floating-point number into mantissa and power

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  frexp(double val, int *exp);
```

[Return value]

Returns mantissa m.
frexpf sets 0 to *exp and returns 0 if *val* is 0.
If *val* is ±∞, frexpf returns zero and sets macro EDOM to global variable errno.

[Description]

This function expresses *val* of double type as mantissa m and the pth power of 2. The resulting mantissa m is 0.5 <= | x | < 1.0, unless *val* is zero. p is stored in *exp*. m and p are calculated so that *val* = m * 2^p.

ldexpf

Convert floating-point number to power

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   ldexpf(float val, int exp);
```

[Return value]

Returns the value calculated by*val* x 2^{exp}.

If an underflow or overflow occurs as a result of executing ldexpf, macro ERANGE is set to global variable errno. If an underflow occurs, ldexpf returns an denormal number. If an overflow occurs, it returns HUGE_VAL.

[Description]

This function calculates *val* x 2^{exp}.

ldexp

Convert floating-point number to power

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  ldexp(double val, int exp);
```

[Return value]

Returns the value calculated by*val* x 2^{exp}.

If an underflow or overflow occurs as a result of executing ldexpf, macro ERANGE is set to global variable errno. If an underflow occurs, ldexpf returns an denormal number. If an overflow occurs, it returns HUGE_VAL.

[Description]

This function calculates *val* x 2^{exp}.

modff

Divide floating-point number into integer and decimal

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   modff(float val, float *ipart);
```

[Return value]

Returns a decimal part. The sign of the result is the same as the sign of *val*.

[Description]

This function divides *val* of float type into integer and decimal parts, and stores the integer part in **ipart*. Rounding is not performed. It is guaranteed that the sum of the integer part and decimal part accurately coincides with *val*. For example, where *realpart* = modff (*val*, &*intpart*), "*realpart* + *intpart*t" coincides with *val*.

modf

Divide floating-point number into integer and decimal

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  modf(double val, double *ipart);
```

[Return value]

Returns a decimal part. The sign of the result is the same as the sign of *val*.

[Description]

This function divides *val* of double type into integer and decimal parts, and stores the integer part in **ipart*. Rounding is not performed. It is guaranteed that the sum of the integer part and decimal part accurately coincides with *val*. For example, where *realpart* = modff (*val*, &*intpart*), "*realpart* + *intpart*t" coincides with *val*.

cosf

Cosine

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   cosf(float x);
```

[Return value]

Returns the cosine of *x*.
If inputting ±∞, cosf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

cos

Cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  cos(double x);
```

[Return value]

Returns the cosine of *x*.
If inputting ±∞, cos returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

sinf

Sine

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   sinf(float x);
```

[Return value]

Returns the sine of *x*.
If inputting ±∞, sinf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, sinf sets macro ERANGE to global variable errno.

[Description]

This function calculates the sine of *x*. Specify the angle in radian.

sin

Sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  sin(double x);
```

[Return value]

Returns the sine of *x*.
If inputting ±∞, sin returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, sin sets macro ERANGE to global variable errno.

[Description]

This function calculates the sine of *x*. Specify the angle in radian.

tanf

Tangent

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   tanf(float x);
```

[Return value]

Returns the tangent of *x*.
If inputting ±∞, tanf returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, tanf sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

tan

Tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  tan(double x);
```

[Return value]

Returns the tangent of *x*.
If inputting ±∞, tan returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno.
If the solution is a denormal number, tan sets macro ERANGE to global variable errno.

[Description]

This function calculates the cosine of *x*. Specify the angle in radian.

acosf

Arc cosine

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   acosf(float x);
```

[Return value]

Returns the arc cosine of *x*. The returned value is in radian and in a range of 0 to π.
If *x* is not between -1 and 1, a Not a Nuber(NaN) is returned, and macro EDOM is set to global variable errno.

[Description]

This function calculates the arc cosine of *x*. Specify *x* as, -1<= *x* <= 1.

acos

Arc cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  acos(double x);
```

[Return value]

Returns the arc cosine of *x*. The returned value is in radian and in a range of 0 to π.
If *x* is not between -1 and 1, a Not a Nuber(NaN) is returned, and macro EDOM is set to global variable errno.

[Description]

This function calculates the arc cosine of *x*. Specify *x* as, -1<= *x* <= 1.

asinf

Arc sine

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   asinf(float x);
```

[Return value]

Returns the arc sine (arcsine) of *x*. The returned value is in radian and in a range of -π / 2 to π / 2.
If *x* is not between -1 and 1, a Not a Nuber(NaN) is returned, and macro EDOM is set to global variable errno.

[Description]

This function calculates the arc sine (arcsine) of *x*. Specify *x* as, -1 <= *x* <= 1.

asin

Arc sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  asin(double x);
```

[Return value]

Returns the arc sine (arcsine) of *x*. The returned value is in radian and in a range of -π / 2 to π / 2.
If *x* is not between -1 and 1, a Not a Nuber(NaN) is returned, and macro EDOM is set to global variable errno.

[Description]

This function calculates the arc sine (arcsine) of *x*. Specify *x* as, -1 <= *x* <= 1.

atanf

Arc tangent

[Classification]

Mathematical library

[Syntax]

#include <mathf.h>
float atanf(float *x*);

[Return value]

Returns the arc tangent (arctangent) of *x*. The returned value is in radian and in a range of -p / 2 to p / 2.
If the solution is a denormal number, atanf sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent (arctangent) of *x*. Specify *x* as, -1 <= *x* <= 1.

[Example]

```
#include    <mathf.h>
float func(float x) {
        float   ret;
        ret = atanf(x); /*Returns value of arctangent of x to ret.*/
          :
        return(ret);
}
```

atan

Arc tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  atan(double x);
```

[Return value]

Returns the arc tangent (arctangent) of *x*. The returned value is in radian and in a range of -π / 2 to π / 2.
If the solution is a denormal number, atan sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent (arctangent) of *x*. Specify *x* as, -1 <= *x* <= 1.

atan2f

Arc tangent (y / x)

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   atan2f(float y, float x);
```

[Return value]

Returns the arc tangent (arctangent) of *y* / *x*. The returned value is in radian and in a range of -π to π.

atan2f returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if both *x* and *y* are 0.0. If the solution vanished approaching zero, atan2f returns ±0 and sets macro ERANGE to global variable errno. If the solution is a denormal number, atan2f sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent of *y* / *x*. atan2f calculates the correct result even if the angle is in the vicinity of π / 2 or - π / 2 (if *x* is close to 0).

atan2

Arc tangent (y / x)

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  atan2(double y, double x);
```

[Return value]

Returns the arc tangent (arctangent) of *y* / *x*. The returned value is in radian and in a range of -π to π.

atan2f returns a Not a Nuber(NaN) and sets macro EDOM to global variable errno if both *x* and *y* are 0.0. If the solution vanished approaching zero, atan2f returns ±0 and sets macro ERANGE to global variable errno. If the solution is a denormal number, atan2f sets macro ERANGE to global variable errno.

[Description]

This function calculates the arc tangent of *y* / *x*. atan2f calculates the correct result even if the angle is in the vicinity of π / 2 or - π / 2 (if *x* is close to 0).

coshf

Hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

#include <mathf.h>
float coshf(float *x*);

[Return value]

Returns the hyperbolic cosine of *x*.
coshf returns HUGE_VAL and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic cosine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} + e^{-x}) / 2$

[Example]

```
#include    <mathf.h>
float func(float x) {
        float   ret;
        ret = coshf(x);     /*Returns value of hyperbolic cosine of x to ret.*/
          :
        return(ret);
}
```

cosh

Hyperbolic cosine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  cosh(double x);
```

[Return value]

Returns the hyperbolic cosine of *x*.
coshf returns HUGE_VAL and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic cosine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} + e^{-x}) / 2$

sinhf

Hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <mathf.h>
float   sinhf(float x);
```

[Return value]

Returns the hyperbolic sine of *x*.
sinhf returns HUGE_VAL and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic sine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} - e^{-x}) / 2$

sinh

Hyperbolic sine

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  sinh(double x);
```

[Return value]

Returns the hyperbolic sine of *x*.
sinhf returns HUGE_VAL and sets macro ERANGE to global variable errno if an overflow occurs.

[Description]

This function calculates the hyperbolic sine of *x*. Specify the angle in radian. The definition expression is as follows.

$(e^{x} - e^{-x}) / 2$

tanhf

Hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
float   tanhf(float x);
```

[Return value]

Returns the hyperbolic tangent of *x*.
If the solution is a denormal number, tanhf sets macro ERANGE to global variable errno.

[Description]

This function calculates the hyperbolic tangent of *x*. Specify the angle in radian. The definition expression is as follows.

sinh (*x*) / cosh (*x*)

tanh

Hyperbolic tangent

[Classification]

Mathematical library

[Syntax]

```
#include <math.h>
double  tanh(double x);
```

[Return value]

Returns the hyperbolic tangent of *x*.
If the solution is a denormal number, tanh sets macro ERANGE to global variable errno.

[Description]

This function calculates the hyperbolic tangent of *x*. Specify the angle in radian. The definition expression is as follows.

sinh (*x*) / cosh (*x*)

7.4.11 RAM section initialization function

RAM section initialization function are as follows.

Table 7.13 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 7.14 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 7.15 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

8. STARTUP

This chapter explains the startup.

8.1 Outline

The startup processing is used to initialize a section for embedding the user application described with the C language to the system or start the main function.

This section assumes two types of programs: a program for a single core device which uses only the single core, and a program for a multi-core device which uses multiple cores.

The program for a single core device uses one application project only.

The program for a multi-core device provides one boot loader project and the required number of application projects.

The following shows the configuration of the basic startup routine to operate those programs, using examples of R7F701Z07 and R7F7010352AFP.

8.2 Startup Routine

Startup routine is the routine that is to be executed after microcontroller is reset and before the execution of main function. Basically, it carries out the initialization after system is reset.

Here describes the following:

- Exception vector table
- Startup routine for the boot loader project
- Startup routine for the application project
- Passing information from the application project to the boot loader project

8.2.1 Exception vector table

A branch destination for the case of reset of the microcontroller or generation of other exceptions is specified for the exception vector table.

The exception vector table is described in vecttbl.asm in the sample program and referenced from the following projects.

For a program for a single core device: application project
For a program for a multi-core device: boot loader project

vecttbl.asm is configured by the following elements.

- RESET vector
- Interrupt handler table
- Exception handler routine

(1) RESET vector

A branch instruction to the entry point routine, which is common to the processor element (PE), is allocated to the address where a branch of the program counter occurs when the microcontroller is started and reset.

In the sample program, the branch instruction is allocated to the RESET section. The section name is optionally changeable but needed be changed in conjunction with the -start option of the optimizing linker.

```
-start=RESET/01000000
```

Caution For the address of the RESET vectors, see the user's manual of the device.

The EBASE register is handled that holds the same value as the RBASE register and shares the exception vector table including the RESET vector. A branch instruction to exception handler addresses of the standard specification (direct vector method) is allocated in the RESET section.

When the RBASE register is operated similarly with the EBASE register regardless of the state of the PSW.EBV register, the following description is required in the startup routine.

```
stsr    2, r10, 1       ; get RBASE
ldsr    r10, 3, 1       ; set EBASE
```

Remark The initial state of the device is PSW.EBV = 0; it is not necessary because EBASE is not used.

The RBASE and EBASE registers, which hold addresses, hold an address in the unit of 512 bytes; the top of the RESET section is aligned at the 512-byte boundary.

Caution For the alignment address or the number of vectors of the RESET section, see the user's manual of the device.

(2) Interrupt handler table
When an exception handler address of the extended specification (table lookup method) is used, the address of the exception handler routine is allocated to the corresponding element position on this table.
In the sample program, the address is allocated to the EIINTTBL section. The section name is optionally changeable but needed be changed in conjunction with the processing of the INTBP register setting in the startup.

```
mov     #__sEIINTTBL, r10
ldsr    r10, 4, 1               ; set INTBP
```

The INTBP register, which holds table addresses, holds an address in the unit of 512 bytes; the top of the EIINTTBL section is aligned at the 512-byte boundary.

Caution For the maximum number of elements in the table, see the user's manual of the device.

(3) Exception handler routine
The sample program for the exception handler routine of FE and EI levels repeats branches to itself without any operation. Usually, it uses #pragma interrupt with the C source description.

8.2.2 Startup routine for the boot loader project

The startup routine for the boot loader project is used in a program for a multi-core device and configured with two files: boot.asm and vecttbl.asm (Exception vector table).

boot.asm is configured by the following elements.

- Processing required for a program for a multi-core device
  - Entry routine common to PE
  - __exit routine
  - Entry routine for PE1
  - Entry routine for PE3
- Processing prepared as required
  - hdwinit_PE1 routine
  - hdwinit_PE3 routine
  - zeroclr4 routine
  - init_eiint routine

(1) Entry routine common to PE
Each PE commonly executes processing. The PE reads its processor element number (PEID) and branches to entry routines prepared for each PEID. If the PE has no application, the branches converge to the __exit routine just below the entry routine.

(2) __exit routine
The __exit routine repeats branches to itself to put PEs that is not used to sleep.

(3) Entry routine for PE1
Branches to the application project for PE1 are performed with the following procedure.

(a) Call the hardware initialization routine for PE1, hdwinit_PE1().

(b) Call the EI-level exception setting routine, init_eiint(). It is disabled in the sample program; enable the USE_TABLE_REFERENCE_METHOD macro at the top of the file.

Remark Refer to the description of hdwinit_PE1 routine and init_eiint routine explained later. If processing is not required, the routine needs not be called.

(c) Set PC via the FEPC register.
Read the address of the entry routine for the application project from the application information table created in the application project.

(d) Branch to the entry routine of the application. Execute the feret instruction to reflect the above FEPC register value to PC and branch to the entry routine in the application.

(4) Entry routine for PE3
The PE3 entry routine has basically same configuration as the PE1 entry routine, however, in the sample program, it ends processing at the branch to __exit routine, not to the application.

(5) hdwinit_PE1 routine
The following procedure is used for one initialization in the hardware and initialization specific to PE1.

(a) Initialize Global RAM and Local RMA (PE1) for the ECC function. In the sample program, the address range for R7F701Z07 is specified and initialized via Local RAM (self).

Caution For the RAM addresses to be initialized, see the user's manual of the device.

(b) Wait for the end of initialization of PE3. The following shows the procedure.
The mutual exclusion variable (MEV) is used as the flag for waiting processing. In boot.asm, MEV is operated by a macro "MEV_ADDR" which has the MEV address as a value.

- The mutual exclusion variable (MEV) is prepared and initialized by 0.
- Set 1 to bit 0 in MEV which indicates the end of processing of PE1 itself.
- Check the value of bit 1 in MEV which indicates the end of processing of PE3.
- If bit 1 in MEV is 1, the initialization processing of PE1 and PE3 has ended and the program exits the wait processing. If bit 1 is 0, PE3 is in the initialization processing and the program returns to the second processing to continue the wait processing.

Remark 1. If synchronization is not needed in PE1 and PE3, this processing is not required.

Remark 2. The countermeasure for ECC error is taken by continuing the processing for setting 1 to bit 0 after MEV is initialized to 0. The Global RAM for allocating MEV needs not be initialized because it is initialized by PE1 itself, however, initialization is performed to keep the symmetry with PE3.

(6) hdwinit_PE3 routine
The following procedure is used for initialization specific to PE3.

(a) Initialize Local RMA (PE3) for the ECC function. In the sample program, the address range for R7F701Z07 is specified and initialized via Local RAM (self).

Caution For the RAM addresses to be initialized, see the user's manual of the device.

(b) Wait for the end of initialization of PE1. The following shows the procedure.

- Similarly to PE1, the mutual exclusion variable (MEV) is initialized to 0.
- Set 1 to bit 1 in MEV which indicates the end of processing of PE3 itself.
- Check the value of bit 0 in MEV which indicates the end of processing of PE1.
- If bit 0 in MEV is 1, the initialization processing of PE1 and PE3 has ended and the program exits the wait processing. If bit 0 is 0, PE1 is in the initialization processing and the program returns to the second processing to continue the wait processing.

(7) zeroclr4 routine
The zerocir4 routine has the ECC function to initialize each RAM. The address ranges passed from the r6 and r7 registers are cleared to 0.

Caution For the memory to be initialized and the initialization method, see the user's manual of the device.

(8) init_eiint routine
The init_eiint routine is a subroutine that initializes the EI-level exception. The following processing is performed.

- Set the start address of the EIINTTBL section to the INTBP register.

- In the sample program, use the interrupt control register to set the branch methods for some EI-level exceptions to the extended specification (table lookup method). If the exception handler for the extended specification is not used, the setting is not required.

It is disabled in the sample program; enable the USE_TABLE_REFERENCE_METHOD macro at the top of the file.

Caution For the interrupt control register, see the user's manual of the device

8.2.3 Startup routine for the application project

The startup routine for the application project sets initialization in the unit of application required for executing the user application (after the main function).

- In the program for a single core device, two files (vecttbl.asm and cstart.asm) are prepared.
- In the program for a multi-core device, cstart.asm is prepared for each application project.
  vecttbl.asm is only prepared for the boot loader project.

vecttbl.asm (Exception vector table) is configured by the following elements.

- RESET vector
- Interrupt handler table
- Exception handler routine

Remark For setting information other than that passed from the application project to the boot loader project, both of label names and section names can be overlapped between application projects. To avoid confusion due to overlapping, when a section name specific to each project is used, specify the -Xmulti_level=1 option of the compiler and use #pragma pmodule on the C source program.

cstart.asm is configured by the following elements.

- Stack area
- RAM section initialization table
- Entry routine
- Processing routine branching to the user application
- abort routine
- hdwinit routine
- zeroclr4 routine
- init_eiint routine
- Application information table

(1) Stack area
The stack area is used for the compiler generation code.
In the sample program, the stack area is reserved for the .stack.bss section of the bss attribute.

Remark When the stack area is reserved for other than the .bss section, there are two purposes: one is that the user variable area and the allocated position are distinguished, and the other is that the program startup time is reduced because initialization can be omitted in the C language specification.

When the memory is referenced by stack pointer (sp) relative in CC-RH, a code is output assuming that sp is allocated in the 4-byte boundary and the stack area is made to be grown to the direction of the 0x0 address. Therefore, the edge of the 0xffffffff address side of the .stack.bss section must be aligned at the 4-byte boundary.

(2) RAM section initialization table
When an external variable is used in the C source program, the area must be initialized before executing the program.
Initialization is performed in the unit of section where external variables are allocated, and the start and end labels of the section generated by a optimizing linker
When the section has the initial value, a line is configured as "#__s section name, #__e section name, #__s initialization-destination section name", as shown below, and the lines are allocated for the required number of sections.

There are following rules in the initialization table for the section with the initial value.

- The initialization table must be allocated to the .INIT_DSEC.const section, and the section name is fixed.
- Only the initialization table can be allocated.
- The start of the initialization table must be aligned at the 4-byte boundary.

```
.section    ".INIT_DSEC.const", const
.align      4
.dw         #__s.data, #__e.data, #__s.data.R
.dw         #__s.sdata, #__e.sdata, #__s.sdata.R
```

To generate the initialization-destination section, the -rom option of the optimizing linker is used.

```
-rom=.data=.data.R
-rom=.sdata=.sdata.R
```

When the section without initial value, a line is configured as "#__s section name, #__e section name", as shown below, and the lines are allocated for the required number of sections.
There are following rules in the initialization table for the section without initial value.

- The initialization table must be allocated to the .INIT_BSEC.const section, and the section name is fixed.
- Only the initialization table can be allocated.
- The start of the initialization table must be aligned at the 4-byte boundary.

```
.section    ".INIT_BSEC.const", const
.align      4
.dw         #__s.bss, #__e.bss
.dw         #__s.sbss, #__e.sbss
```

When the sections described in those tables do not exist in the user application, an error occurs at linking. To avoid this, the empty dummy section as shown below is prepared for the section name described in the table.

```
        .section    ".data", data
.L.dummy.data:
        .section    ".sdata", sdata
.L.dummy.sdata:
        .section    ".bss", bss
.L.dummy.bss:
        .section    ".sbss", sbss
.L.dummy.sbss:
```

When #pragma pmodule is used in the program for a multi-core device, the section name is automatically modified and all the section names that can be appeared are described.

```
        .section    ".INIT_DSEC.const", const
        .align      4
        .dw         #__s.data.pm1, #__e.data.pm1, #__s.data.pm1.R
        .dw         #__s.data.cmn, #__e.data.cmn, #__s.data.cmn.R
        .dw         #__s.data, #__e.data, #__s.data.R
        .dw         #__s.sdata.pm1, #__e.sdata.pm1, #__s.sdata.pm1.R
        .dw         #__s.sdata.cmn, #__e.sdata.cmn, #__s.sdata.cmn.R
```

```
        .dw         #__s.sdata, #__e.sdata, #__s.sdata.R

        .section    ".INIT_BSEC.const", const
        .align      4
        .dw         #__s.bss.pm1, #__e.bss.pm1
        .dw         #__s.bss.cmn, #__e.bss.cmn
        .dw         #__s.bss, #__e.bss
        .dw         #__s.sbss.pm1, #__e.sbss.pm1
        .dw         #__s.sbss.cmn, #__e.sbss.cmn
        .dw         #__s.sbss, #__e.sbss

        .section    ".data.pm1", data
.L.dummy.data.pm1:
        .section    ".data.cmn", data
.L.dummy.data.cmn:
        .section    ".data", data
.L.dummy.data:
        .section    ".sdata.pm1", sdata

.L.dummy.data.pm1:
        .section    ".sdata.cmn", sdata
.L.dummy.data.cmn:
        .section    ".sdata", sdata
.L.dummy.data:

        .section    ".bss.pm1", bss
.L.dummy.bss.pm1:
        .section    ".bss.cmn", bss
.L.dummy.bss.cmn:
        .section    ".bss", bss
.L.dummy.bss:

        .section    ".sbss.pm1", sbss
.L.dummy.bss.pm1:
        .section    ".sbss.cmn", sbss
.L.dummy.bss.cmn:
        .section    ".sbss", sbss
.L.dummy.sbss:
```

Remark When the contents of the .INIT_DSEC.const and .INIT_BSEC.const sections are sufficient, no dummy sections are required.

(3) Entry routine
The execution environment (PE) is initialized with the following procedure.

(a) Set the most significant address of the stack area to the sp (r3) register. The set value must be a multiple of 4.

(b) Set the address of the __gp_data label to the gp (r4) register. The __gp_data label is automatically generated by a optimizing linker when the gp relative section is in use.

(c) Set the address of the __ep_data label to the ep (r30) register. The __ep_data label is automatically generated by a optimizing linker when the ep relative section is in use.

Remark When the gp relative and ep relative sections are not in use in the program, even if there are setting codes for gp and ep, it is only redundant and no problem occurs.

(d) Call hdwinit() and make specific initialization to PE.
The same initialization is allowed in the boot loader project; initialization based on the data defined in the application side is performed in the application project side, and initialization by the data shared between PEs is performed in the boot loader project side.

(e) Call __INITSCT_RH to initialize the RAM section.

- When both sections with or without initial values are initialized:

```
mov     #__s.INIT_DSEC.const, r6
mov     #__e.INIT_DSEC.const, r7
mov     #__s.INIT_BSEC.const, r8
mov     #__e.INIT_BSEC.const, r9
jarl    __INITSCT_RH, lp
```

- When only the section with initial values is initialized:

```
mov     #__s.INIT_DSEC.const, r6
mov     #__e.INIT_DSEC.const, r7
mov     r0, r8
mov     r0, r9
jarl    __INITSCT_RH, lp
```

- When only the section without initial values is initialized:

```
mov     r0, r6
mov     r0, r7
mov     #__s.INIT_BSEC.const, r8
mov     #__e.INIT_BSEC.const, r9
jarl    __INITSCT_RH, lp
```

(f) Branch to the processing that branches to the user application.

(4) Processing routine branching to the user application
Branches to the user application are performed with the following procedure.

(a) Set PSW and the branch destination to branch to the user application via the FEPSW register.
In the sample program, the following settings are made.

Target Program	FPU	EI-level Exception	User Mode
Program for a single core device (for R7F7010352AFP)	OFF	OFF	OFF
Program for a multi-core device (for R7F701Z07)	ON	OFF	ON

(b) Set the start address (#_main) of the user application to the FEPC register.

(c) Execute the feret instruction to reflect the PSW setting and branch to the user application.

Remark When the operating mode of the CPU is not changed, a branch instruction, not the feret instruction, is available.

```
jarl32  #_main, lp
```

(5) abort routine
The abort routine is required when the assert macro or abort itself is used in the user application. In the sample program, branching to the abort routine itself is only repeated.

(6) hdwinit routine
In the sample program for a single core device, a Local RAM is initialized for the ECC function.

Caution The sample program specifies the address range for R7F7010352AFP.
See the user's manual of the device and specify the appropriate address.

(7) zeroclr4 routine
The zerocir4 routine has the ECC function to initialize each RAM. The address ranges passed from the r6 and r7 registers are cleared to 0.

Caution For the memory to be initialized and the initialization method, see the user's manual of the device.

(8) init_eiint routine
The init_eiint routine is a subroutine that initializes the EI-level exception. The following processing is performed.

- Set the start address of the EIINTTBL section to the INTBP register.
- In the sample program, use the interrupt control register to set the branch methods for some EI-level exceptions to the extended specification (table lookup method). If the exception handler for the extended specification is not used, the setting is not required.

It is disabled in the sample program; enable the USE_TABLE_REFERENCE_METHOD macro at the top of the file.

Caution For the interrupt control register, see the user's manual of the device.

(9) Application information table
The application information table stores context information that is passed to the boot loader project in the program for a multi-core device.
In the sample program, the following information is stored in the table.

Offset	Value	Remark
0	Entry routine address of the application	Specifies the absolute address of the label corresponding to __start

The section name for allocating a table is .const.cmn. The label in this section is referenced from the boot loader project.

Caution When the startup routine is copied between application projects, the label names in .const.cmn must be changed so that they are not overlapped.

Remark The application information table can be changed as required. To change this table, the table in the boot loader project side must also be read.

8.2.4 Passing information from the application project to the boot loader project

A program for a multi-core device is configured of the boot loader project and multiple application projects.
The -fsymbol option of the optimizing linker is used in the boot loader project to initialize hardware or use the information set in each application project. Accordingly, the public label name in the specified section and information of the address value after linking can be output to the file (symbol address file) using an option and passed to the boot loader project.
The following shows an example for passing the entry routine address from the application project to the boot loader project.

To reference the address of __start_pm1 from the boot loader project, specify __start_pm1 with public.

```
;; cstart.asm
        .section    ".text.cmn", text
        .public     __start_pm1
__start_pm1:
        jarl        _main, lp
```

When the application project is linked, the -fsymbol option specifies the .text.cmn section where the __start_pm1 label exists. In this case, the pm1.fsy file is generated.

```
>rlink cstart.obj .opm1.abs -fsymbol=.text.cmn
```

The boot loader project references __start_pm1.

```
;; boot.asm
        .section    ".text", text
__start:
        cmp         1, r6
        bnz         .L1
        jr32        #__start_pm1
.L1:
```

When the fsy file generated from each application project is compiled with the boot loader project, the address value in the .fsy file resolves references in boot.asm.

```
>ccrh boot.asm pm1.fsy pm2.fsy -oboot.abs
```

Caution All the public labels in the section specified with the -fsymbol option are considered to have been defined in the boot loader project. Therefore, if there are labels with the same name in the specified sections of different application projects, a multiple definition error will occur at linking in the boot loader side (even if

the section names are the same between applications, they will not collide).
Note that many labels must not be defined in the section specified for the fsymbol option in the application project; if defined, the label name must be the one that will not collide with other application projects.

8.3 Coding Example

The following is an example of exception vector table.

Table 8.1 Examples of exception vector table

```
        .section        "RESET", text
        .align  512
        jr32    __start         ; RESET

        .align  16
        jr32    _Dummy          ; SYSERR

        .align  16
        jr32    _Dummy          ; HVTRAP

        .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/ITLBE/DTLBE

        .align  16
        jr32    _Dummy          ; PIE

        .align  16
        jr32    _Dummy          ; Debug

        .align  16
        jr32    _Dummy          ; MAE

        .align  16
        jr32    _Dummy          ; (R.F.U)

        .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)

        .section        "EIINTTBL", const
        .align  512
        .dw     #_Dummy_EI      ; INT0
        .dw     #_Dummy_EI      ; INT1
        .dw     #_Dummy_EI      ; INT2
        .rept   288 - 3
        .dw     #_Dummy_EI      ; INTn
        .endm

        .section        ".text", text
        .align  2
_Dummy:
        br      _Dummy

_Dummy_EI:
        br      _Dummy_EI
```

The following is an example of startup routine of a project for a single core device.

Table 8.2 Examples of startup routine of a project for a single core device

```
; NOTE : THIS IS A TYPICAL EXAMPLE. (R7F7010352AFP)

        ; example of using eiint as table refrence method
        ;USE_TABLE_REFERENCE_METHOD .set 1

;------------------------------------------------------------------------------
;       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", text
        .public     __start
        .align      2
__start:
        mov         #_stacktop, sp          ; set sp register
        mov         #__gp_data, gp          ; set gp register
        mov         #__ep_data, ep          ; set ep register

        jarl        _hdwinit, lp            ; initialize hardware

        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

$ifdef USE_TABLE_REFERENCE_METHOD
        jarl        _init_eiint, lp         ; initialize exception
$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
        ldsr        r10, 2, 0               ; FEPC <- #_main

        ; apply PSW and call main
        feret
_exit:
        br          _exit                   ; end of program

;-------------------------------------------------------------------------------
;      abort
```

```
;-------------------------------------------------------------------------------
        .public     _abort
        .align      2
_abort:
        br          _abort

;-------------------------------------------------------------------------------
;      target dependence informations (specify values suitable to your system)
;-------------------------------------------------------------------------------
        ; RAM address
        PRIMARY_LOCAL_RAM_ADDR      .set    0xfede0000
        PRIMARY_LOCAL_RAM_END       .set    0xfedfffff

        SECONDARY_LOCAL_RAM_ADDR    .set    0xfedd8000
        SECONDARY_LOCAL_RAM_END     .set    0xfeddffff

        RETENTION_RAM_ADDR          .set    0xfee00000
        RETENTION_RAM_END           .set    0xfee07fff

;-------------------------------------------------------------------------------
;      hdwinit
;-------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit:
        mov         lp, r14                 ; save return address

        mov         PRIMARY_LOCAL_RAM_ADDR, r6
        mov         PRIMARY_LOCAL_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Primary local RAM

        mov         SECONDARY_LOCAL_RAM_ADDR, r6
        mov         SECONDARY_LOCAL_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Secondary local RAM

        mov         RETENTION_RAM_ADDR, r6
        mov         RETENTION_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Retention RAM

        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
;-------------------------------------------------------------------------------
;      init_eiint
;-------------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE                      .set    0xffff9000
        .align      2
_init_eiint:
        mov         #__sEIINTTBL, r10
        ldsr        r10, 4, 1           ; set INTBP
         ; some inetrrupt 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

;-------------------------------------------------------------------------------
;      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 -------------------;
```

The following is an example of startup routine of a boot loader project for a multi-core device.

Table 8.3 Examples of startup routine of a boot loader project for a multi-core device

```
; NOTE  : THIS IS A TYPICAL EXAMPLE. (R7F701Z07)

        ; example of using eiint as table reference method
        ;USE_TABLE_REFERENCE_METHOD .set 1

        ; offset of processing module setting table element
        .OFFSET_ENTRY       .set    0

;-------------------------------------------------------------------------------
;      startup
;-------------------------------------------------------------------------------
        .section    ".text", text
        .public     __start
        .align      2
__start:
        ; jump to entry point of each PE
        stsr        0, r10, 2               ; get HTCFG0
        shr         16, r10                 ; get PEID

        cmp         1, r10
        bz          __start_PE1
        cmp         3, r10
        bz          __start_PE3
__exit:
        br          __exit

__start_PE1:
        jarl        _hdwinit_PE1, lp        ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        jarl        _init_eiint, lp         ; initialize exception
$endif

        mov         #_pm1_setting_table, r13
        ld.w        .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        ldsr        r10, 2, 0               ; set FEPC

        ; apply PSW and jump to the application entry point
        feret

__start_PE3:
        jarl        _hdwinit_PE3, lp        ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
```

```
        jarl        _init_eiint, lp         ; initialize exception
$endif
        ;mov        #_pm2_setting_table, r13
        ;ld.w       .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        ;ldsr       r10, 2, 0               ; set FEPC

        ; apply PSW and jump to the application entry point
        ;feret
        br          __exit

;-------------------------------------------------------------------------------
;      target dependence informations (specify values suitable to your system)
;-------------------------------------------------------------------------------
$if 1
        ; RAM address
        GLOBAL_RAM_ADDR             .set    0xfeef0000
        GLOBAL_RAM_END              .set    0xfef1ffff

        LOCAL_RAM_PE1_ADDR          .set    0xfedf0000
        LOCAL_RAM_PE1_END           .set    0xfedfffff

        LOCAL_RAM_PE3_ADDR          .set    0xfedf8000
        LOCAL_RAM_PE3_END           .set    0xfedfffff

        ; mutual exclusion variable
        MEV_ADDR                    .set    0xfeef0000

;-------------------------------------------------------------------------------
;      hdwinit_PE1
;-------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit_PE1:
        mov         lp, r14                 ; 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

        ; wait for PE3
```

```
        mov         MEV_ADDR, r10
        st.w        r0, [r10]
.L.hdwinit_PE1.1:
        snooze
        set1        0, [r10]
        tst1        1, [r10]
        bz          .L.hdwinit_PE1.1

        mov         r14, lp
        jmp         [lp]

;-------------------------------------------------------------------------------
;      hdwinit_PE3
;-------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit_PE3:
        mov         lp, r14                 ; save return address

        ; clear Local RAM PE3
        mov         LOCAL_RAM_PE3_ADDR, r6
        mov         LOCAL_RAM_PE3_END, r7
        jarl        _zeroclr4, lp

        ; wait for PE1
        mov         MEV_ADDR, r10
        st.w        r0, [r10]
.L.hdwinit_PE3.1:
        snooze
        set1        1, [r10]
        tst1        0, [r10]
        bz          .L.hdwinit_PE3.1

        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]
$endif

$ifdef USE_TABLE_REFERENCE_METHOD
;-------------------------------------------------------------------------------
;      init_eiint
;-------------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE .set 0xfffeea00

        .align 2
_init_eiint:
        mov         #__sEIINTTBL, r10
        ldsr        r10, 4, 1               ; set INTBP


        ; some inetrrupt 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 -------------------;
```

The following is an example of startup routine of an application project for a multi-core device.

Table 8.4 Examples of startup routine of an application project for a multi-core device

```
; NOTE  : THIS IS A TYPICAL EXAMPLE. (R7F701Z07)

;-------------------------------------------------------------------------------
;      processing module setting table
;-------------------------------------------------------------------------------
        .section    ".const.cmn", const
        .public     _pm1_setting_table
        .align      4
_pm1_setting_table:
        .dw         #__start                ; ENTRY ADDRESS

;-------------------------------------------------------------------------------
;      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", text
        .public     __start
        .align      2
__start:
        mov         #_stacktop, sp          ; set sp register
        mov         #__gp_data, gp          ; set gp register
        mov         #__ep_data, ep          ; set ep register

        jarl        _hdwinit, lp            ; initialize hardware

        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                ; enable FPU
```

```
        ;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

;-------------------------------------------------------------------------------
;      abort
;-------------------------------------------------------------------------------
        .public     _abort
        .align      2
_abort:
        br          _abort

;-------------------------------------------------------------------------------
;      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

The CC-RH uses the following pointers for operation of applications.

- Global pointer (gp)
- Element pointer (ep)

This section describes the role of each pointer and how pointer values are determined.

8.4.1 Global pointer (gp)

Data that is globally declared in an application is allocated to memory. When referencing (loading or storing) this data that has been allocated to memory, the global pointer (gp) is provided to enable access independent of the allocation position (PID: Position Independent Data).

(1) Setting the global pointer (gp)
Set the value of the global pointer symbol (gp symbol) as the address set in the global pointer (gp).

- The gp symbol is handled as the constant symbol name "__gp_data".
- Declare the gp symbol as a reference symbol (.extern) in the startup routine.
- rlink creates an externally defined symbol (.public) and determines the address of the gp symbol.
- The operations for defining and referencing the gp symbol are as follows.

(a) If there is only an ".extern" declaration of the gp symbol
rlink creates the definition information and sets the address automatically.

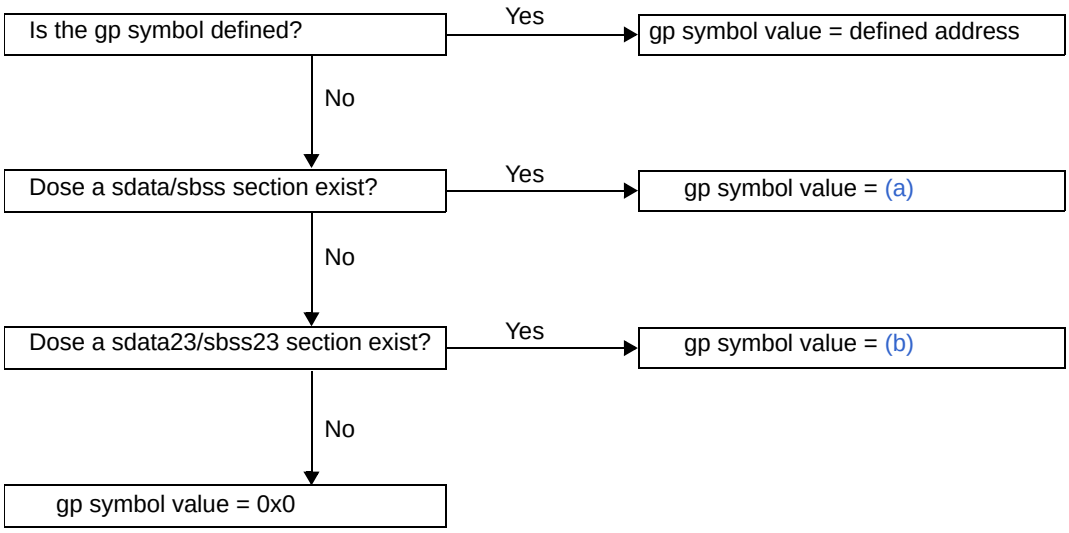
(b) If the gp symbol is defined
The defined address is used.

(c) If there is no gp symbol (it is not used)
The following error message is output for code that makes references relative to the gp.

```
Undefined external symbol "GP-symbol (__gp_data)" referenced in "FILE"
```

(2) Rules for determining the gp symbol value
The value of the gp symbol is determined from sections with relocation attributes of sdata, sbss, sdata23, and sbss23, in the following order of precedence.

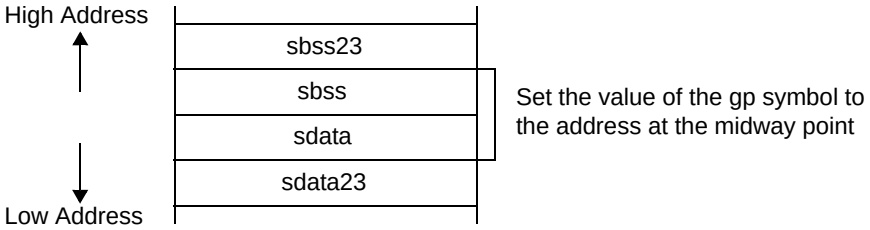
Figure 8.1 Rules for Determining the Global Pointer Value



Below is an example of how the value of the gp symbol is determined.

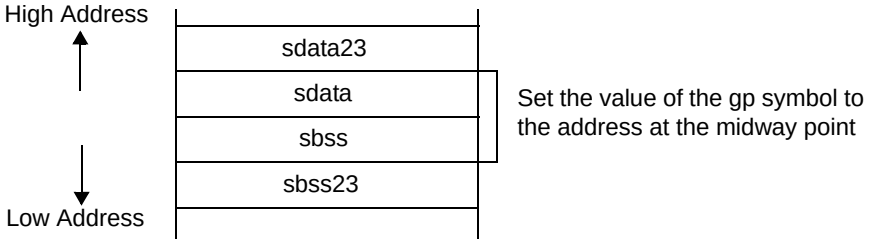
(a) If there is a section with a relocation attribute of sdata or sbss
Sets the halfway point between the start address of the sdata or sbss section with the smallest address, and the end address of the sdata or sbss section with the greatest address (multiple of 2; if the midway point is an odd number, takes the first multiple of two) as the address value of the gp symbol.

<1> If an sdata and sbss section are located in this order, from smallest to highest address



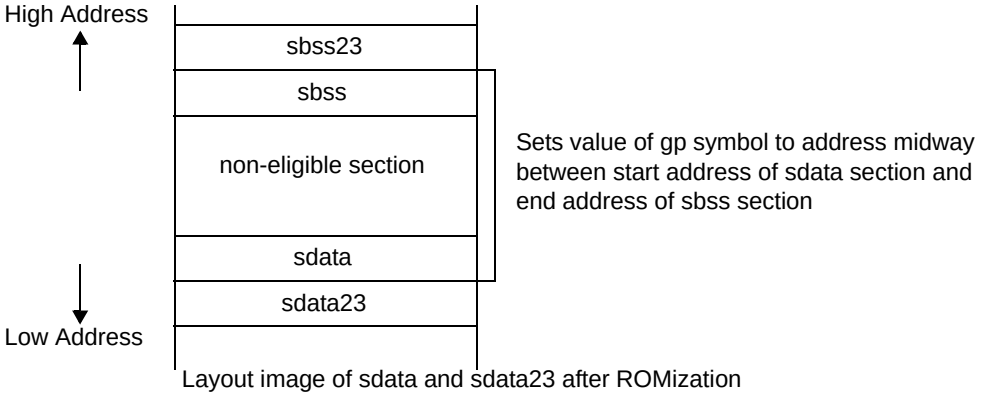
Layout image of sdata and sdata23 after ROMization

<2> If an sbss and sdata section are located in this order, from smallest to highest address



Layout image of sdata and sdata23 after ROMization

<3> If sdata and sbss sections are placed in this order, from lowest to highest address, and there is a non-eligible section in between the sdata and sbss sections

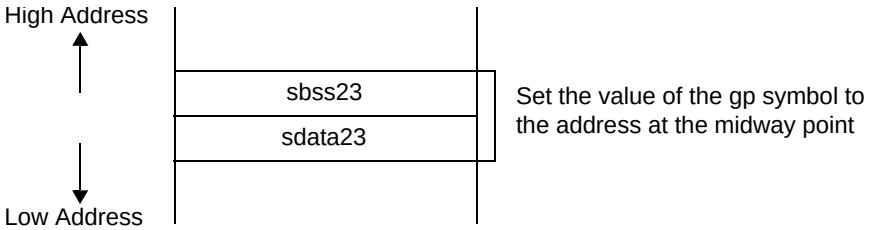


Layout image of sdata and sdata23 after ROMization

(b) If there is a section with a relocation attribute of sdata23 or sbss23

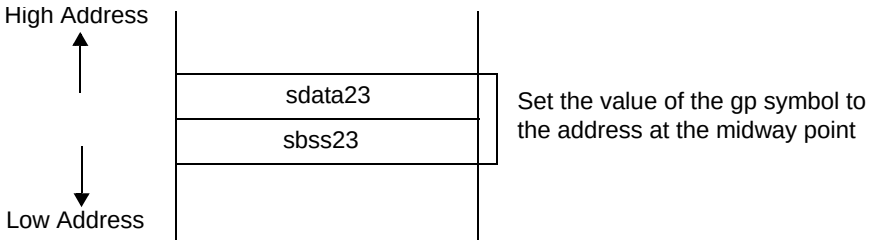
Sets the halfway point between the start address of the sdata23 or sbss23 section with the smallest address, and the end address of the sdata or sbss section with the greatest address (multiple of 2; if the midway point is an odd number, takes the first multiple of two) as the address value of the gp symbol.

<1> If an sdata23 and sbss23 section are located in this order, from smallest to highest address



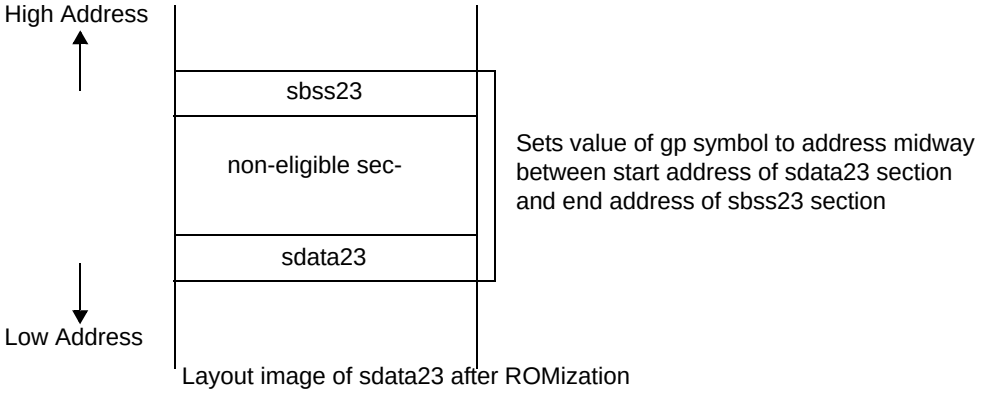
Layout image of sdata23 after ROMization

<2> If an sbss23 and sdata23 section are located in this order, from smallest to highest address



Layout image of sdata23 after ROMization

<3> If sdata23 and sbss23 sections are placed in this order, from lowest to highest address, and there is a non-eligible section in between the sdata23 and sbss23 sections



Layout image of sdata23 after ROMization

(c) If there are no sections with sdata, sbss, sdata23, or sbss23 relocation attributes

If there are no sdata, sbss, sdata23, or sbss23 sections, then the address value of the gp symbol is set to zero (0).

8.4.2 Element pointer (ep)

The element pointer is a pointer that is provided to realize faster access (loading and storing) by allocating data (variables) that are globally declared within an application to RAM area in RH850 device.

Data (variables) that is globally declared and allocated to internal RAM area is referenced with ep-relative.

(1) Setting the element pointer (ep)

Set the value of the element pointer symbol (ep symbol) as the address set in the global pointer (ep).

- The ep symbol is handled as the constant symbol name "__ep_data".
- Declare the ep symbol as a reference symbol (.extern) in the startup routine.
- rlink creates an externally defined symbol (.public) and determines the address of the ep symbol.
- The operations for defining and referencing the ep symbol are as follows.

(a) If there is only an ".extern" declaration of the ep symbol

rlink creates the definition information and sets the address automatically.

(b) If the ep symbol is defined

The defined address is used.

(c) If there is no ep symbol (it is not used)

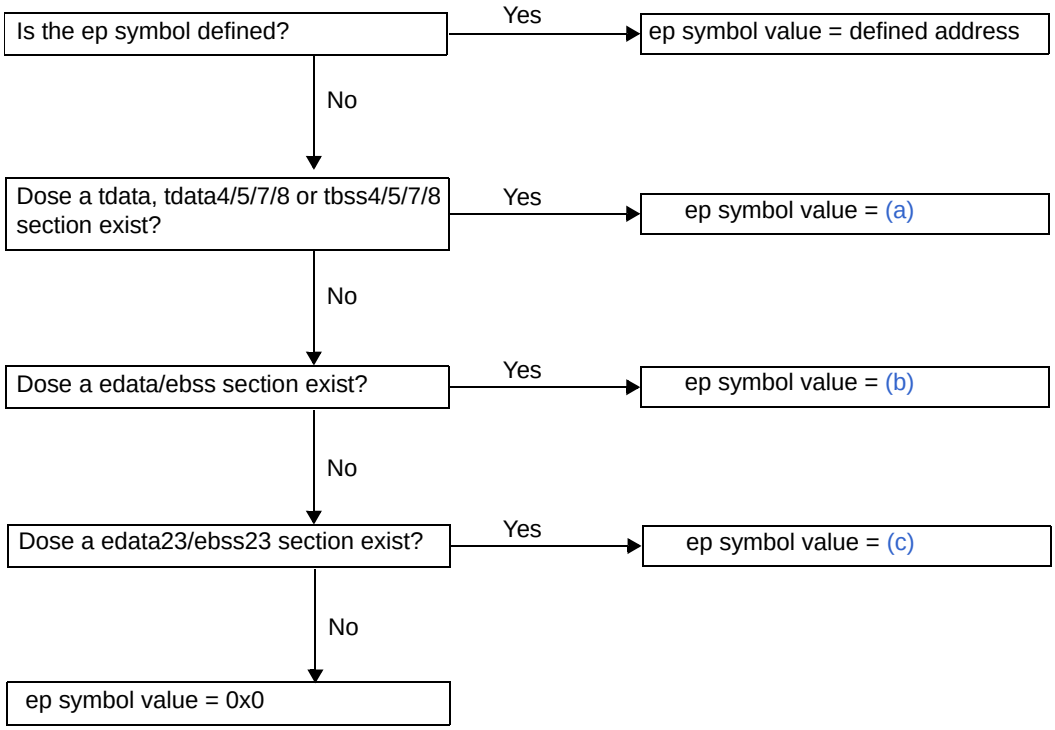
The following error message is output for code that makes references relative to the ep.

```
Undefined external symbol "EP-symbol (__ep_data)" referenced in "FILE"
```

(2) Rules for determining the ep symbol value

The value of the ep symbol is determined from sections with relocation attributes of tdata, tdata4/5/7/8, tbss4/5/7/8, edata, ebss, edata23 and ebss23, in the following order of precedence.

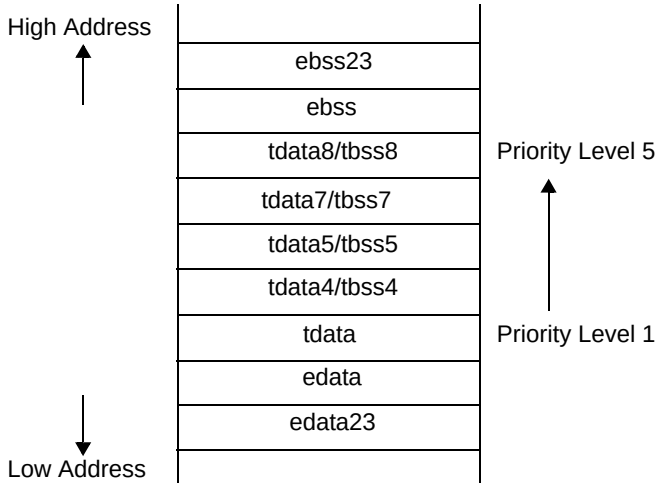
Figure 8.2 Rules for Determining the Element Pointer Value



Below is an example of how the value of the ep symbol is determined.

(a) If there are sections with relocation attributes of tdata, tdata4/5/7/8, and tbss4/5/7/8

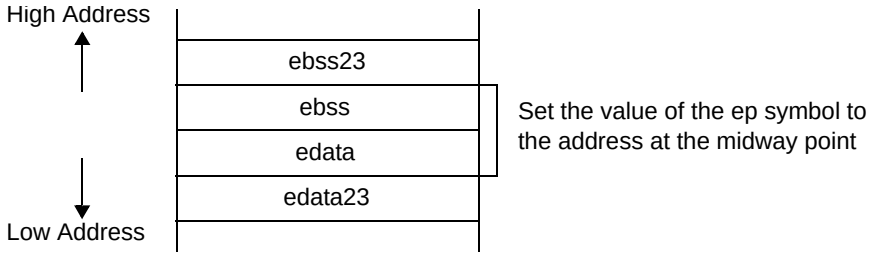
The ep symbol is set to the start address of the first section in this order: tdata -> tdata4, or tbss4 -> ... -> tdata8, or tbss8.



Layout image of tdata, tdata4/5/7/8, edata and edata32 after ROMization

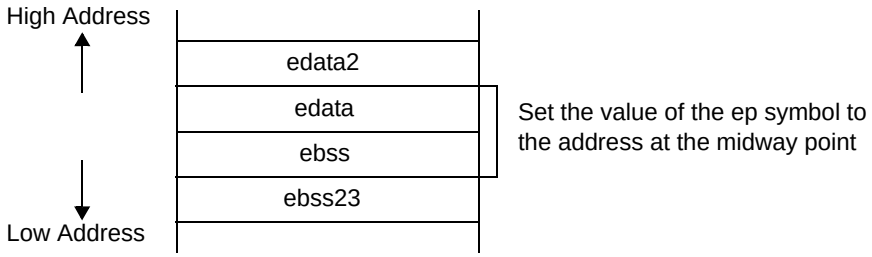
(b) If there is a section with a relocation attribute of edata or ebss
Sets the halfway point between the start address of the edata or ebss section with the smallest address, and the end address of the edata or ebss section with the greatest address (multiple of 2; if the midway point is an odd number, takes the first multiple of two) as the address value of the ep symbol.

<1> If an edata and ebss section are located in this order, from smallest to highest address



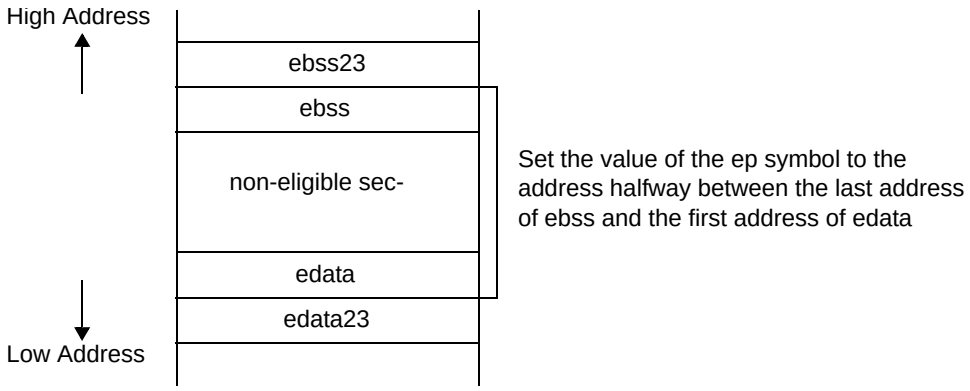
Layout image of edata and edata23 after ROMization

<2> If an ebss and edata section are located in this order, from smallest to highest address



Layout image of edata and edata23 after ROMization

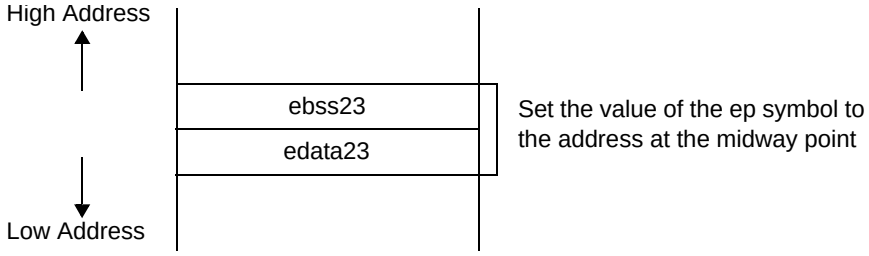
<3> If edata and ebss sections are placed in this order, from lowest to highest address, and there is a non-eligible section in between the edata and ebss sections



Layout image of edata and edata23 after ROMization

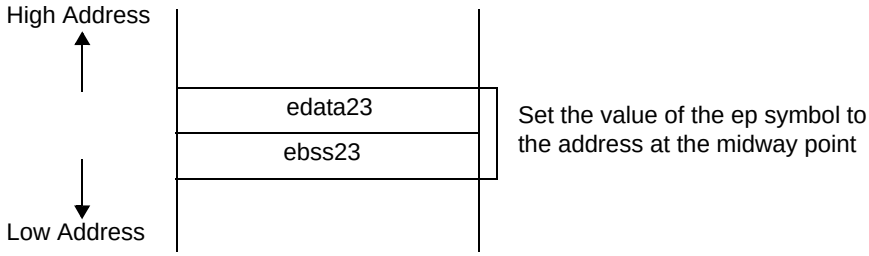
(c) If there is a section with a relocation attribute of edata23 or ebss23
Sets the halfway point between the start address of the edata23 or ebss23 section with the smallest address, and the end address of the edata23 or ebss23 section with the greatest address (multiple of 2; if the midway point is an odd number, takes the first multiple of two) as the address value of the ep symbol.

<1> If an edata23 and ebss23 section are located in this order, from smallest to highest address



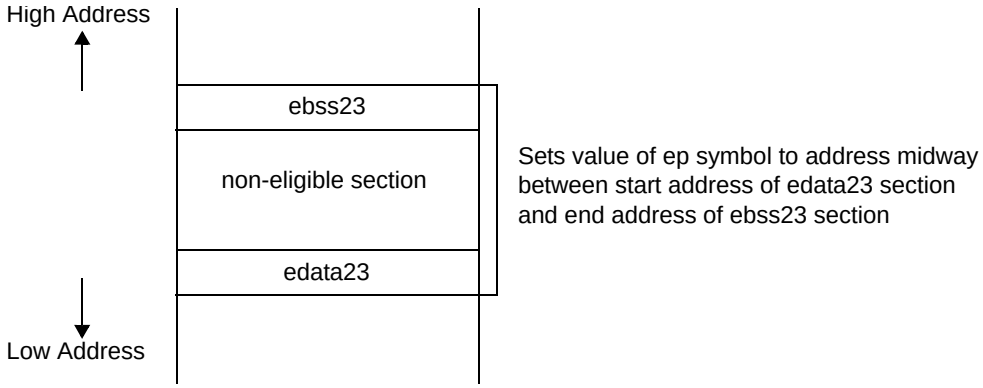
Layout image of edata after ROMization

<2> If an ebss23 and edata23 section are located in this order, from smallest to highest address



Layout image of edata23 after ROMization

<3> If edata23 and ebss23 sections are placed in this order, from lowest to highest address, and there is a non-eligible section in between the edata23 and ebss23 sections



Layout image of edata23 after ROMization

(d) If there are no sections with tdata, tdata4/5/7/8, tbss4/5/7/8, edata, ebss, edata23, or ebss23 relocation attributes

If there are no data, tdata4/5/7/8, tbss4/5/7/8, edata, ebss, edata23, or ebss23 sections, then the address value of the ep symbol is set to zero (0).

8.5 ROMization

This section describes an outline of the ROMization procedure, operation method, etc.

8.5.1 Outline

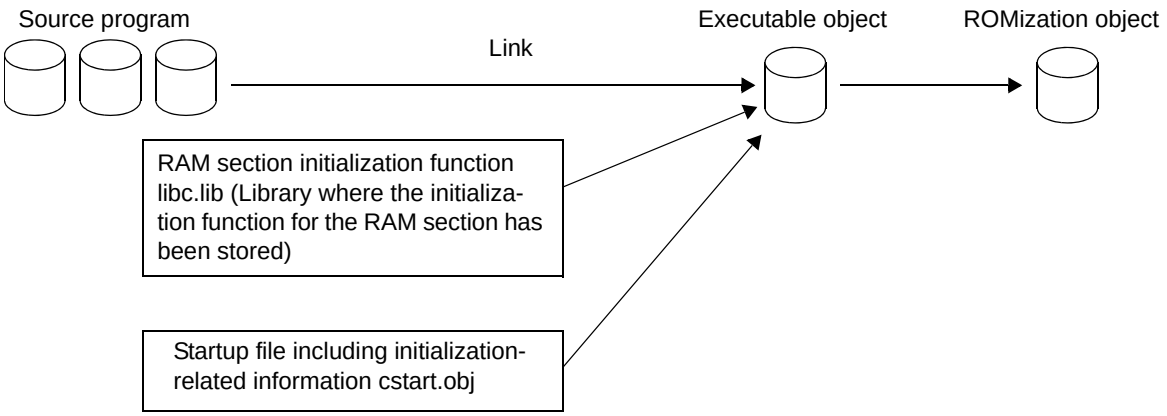
When a variable is declared globally or static within a program, the variable is allocated to the data-attribute section in RAM if the variable has a initial value, or to the bss-attribute section if it does not have a initial value. When the variable has a initial value, that initial value is also stored in RAM. In addition, program code may be stored in the internal RAM area to speed up applications.

In the case of an embedded system, if a debug tool such as an in-circuit emulator is used, executable modules can be downloaded and executed just as they are in the allocation image. However, if the program is actually written to the target system's ROM area before being executed, the initial value information that has been allocated to the data-attribute section and the program code that has been allocated to a RAM area must be deployed in RAM prior to execution. In other words, data that is residing in RAM must be deployed in ROM, and this means that data must be copied from ROM to RAM before the corresponding application is executed.

"ROMization" refers to the packing of the initial values of variables in data-attribute sections and program code to be allocated to the RAM into a single section of ROM. Allocating this section to the ROM and calling the copy function provided by the CC-RH make it easy to deploy the initial values and program code to the RAM.

The following figure shows an outline of the operation flow in creating objects for ROMization.

Figure 8.3 Creation of Object for ROMization



When ROMization objects are created as shown in the "Figure 8.3 Creation of Object for ROMization", execution of the _INITSCT_RH copies the data to be allocated to RAM from the packed ROM section.

The function used to copy from the ROM area to the RAM area is as follows.

- _INITSCT_RH

This function is stored in the library "libc.lib".

If the object files resolved for relocation include symbol information and debug information, the CC-RH creates a ROMization object module file without deleting them. Therefore, the debugger can debug the source even with a ROMization object module file.

8.5.2 Creating ROMized load module file

This section explains how to create the ROMized load module.

The -rom option is used for ROMization. The code is as follows.

```
-rom=name of the initial-value section=name of the destination section[,name of the
initial-value section=name of the destination section]...
```

The -start option must also be used to specify the addresses of the initial-value section and destination section.

```
-start=[(]section-name[{:|,}section-name]...[)][{:|,}section-name]...[/destination-
address][,[(]section-name[{:|,}section-name]...[)][{:|,}section-name]...[/destina-
tion-address]]...
```

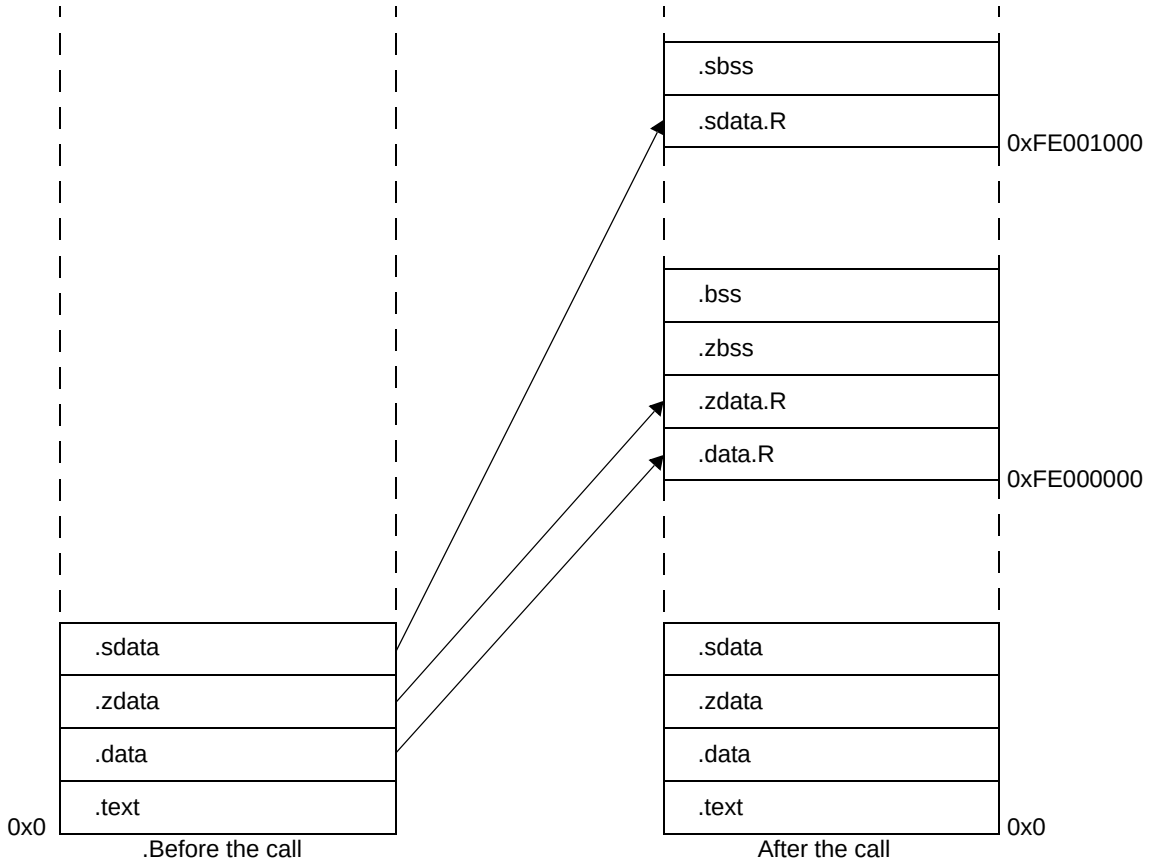
Assume that the program contains seven sections: .text, .data, .zdata, .zbss, .bss, .sdata, and .sbss. If .text, r0 relative sections, and r4 relative sections are to be allocated to addresses 0x0, 0xFE000000, and 0xFE001000 respectively before execution of the user program, the code will be as follows.

```
-start=.text,.data,.zdata,.sdata/0
-start=.data.R,.zdata.R,.zbss,.bss/fe000000
-start=.sdata.R,.sbss/fe001000
-rom=.data=.data.R,.zdata=.zdata.R,.sdata=.sdata.R
```

.data and .zdata are allocated to the range following the .text section at address 0x0.
When the user program is executed, .data.R, .zdata.R, .zbss, and .bss at address 0xFE000000 and .sdata.R and .bss at address 0xFE001000 are initialized before they are used.

An image of this operation is shown below.

Figure 8.4 Image of Processing Before and After Copy Function Call



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. r4 (GP) and r5 (TP) may be locked throughout the entire program. If so, then the rules described here do not apply to them, because the contents of the general-purpose registers are never changed anywhere in the program. The user is responsible for overwriting them.

Remark 5. It is possible to specify usage of r2 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.

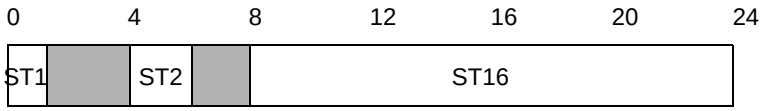
<2> Each argument is essentially aligned on a 4-byte boundary.

<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.

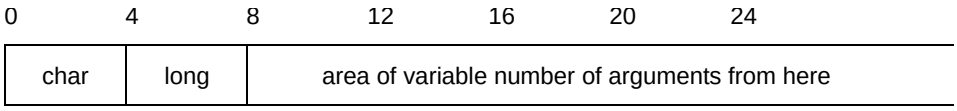
- 1-byte scalar integer ->Promoted to 4-byte integer, then stored
- 2-byte scalar integer ->Promoted to 4-byte integer, then stored
- 4-byte scalar integer ->Stored as-is
- 8-byte scalar integer ->Stored as-is
- 4-byte scalar floating-point number ->Promoted to 8-byte floating-point number, then stored
- 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]



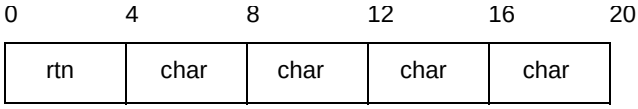
In the case of a structure or union whose size is not a multiple of 4, it is possible to add padding between the parameters. The contents of the padded area are undefined.

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

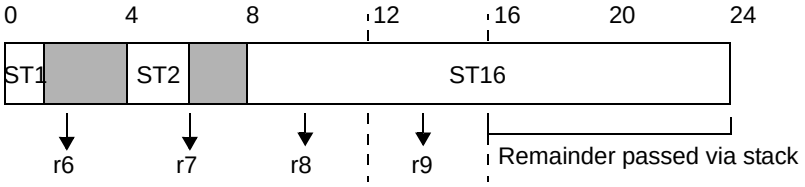
<1> If the arguments passed via the registers, it's loaded by the word units to each register (r6-r9). The byte units and the half-word units aren't loaded.

<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.

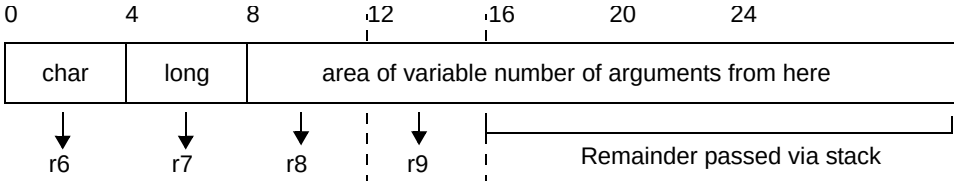
Remark See "9.1.4 Stack frame" about how data is placed on the stack.

Example 1. Function prototype : f(ST1, ST2, ST16)
Example where STx represents a structure with a size of x[byte]



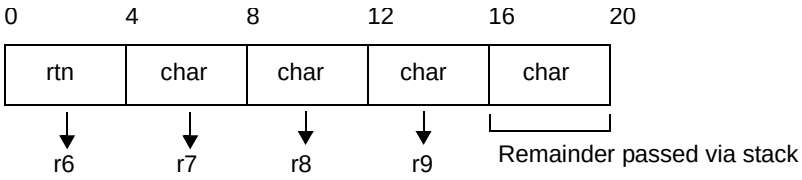
Even if only part of a structure (in this case, ST16) can fit in the registers, that part is still passed via the registers.

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.

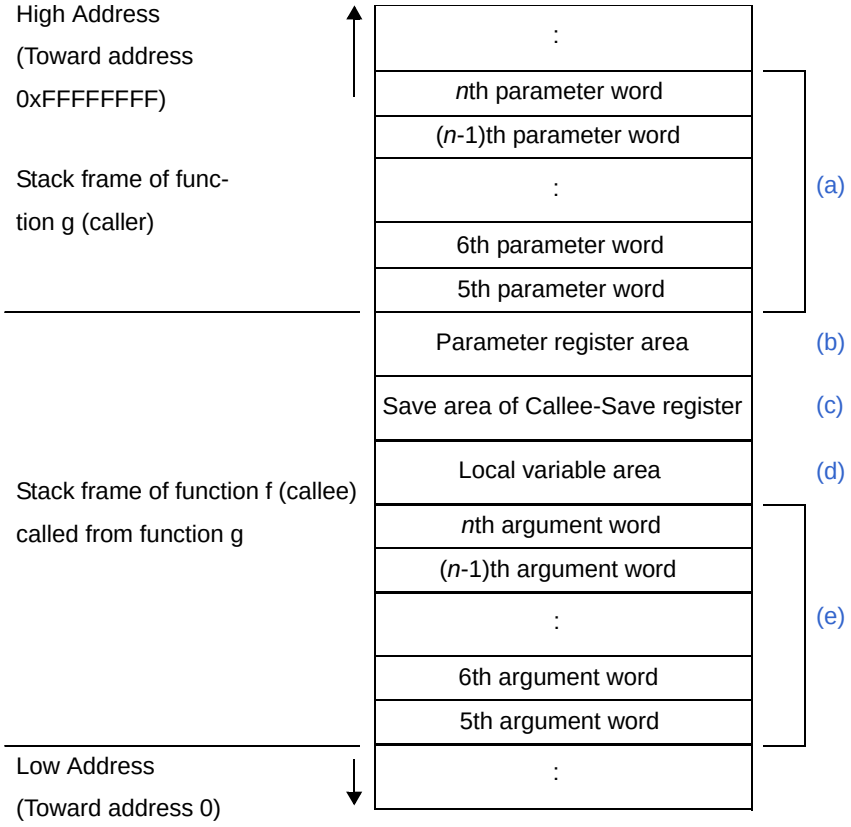
9.1.3 Address indicating stack pointer

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

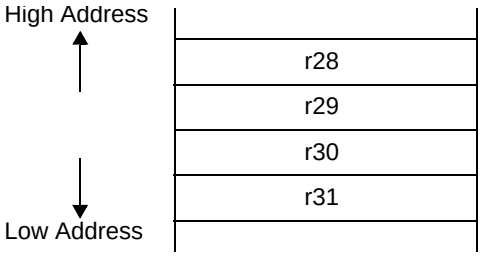


Below is the range of the area that function f can reference and set.

(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.



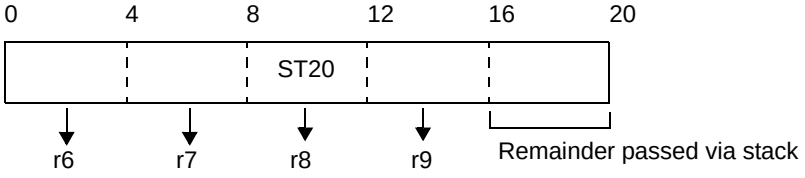
(d) Local variable area
This stack area is used for local variables.

(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.

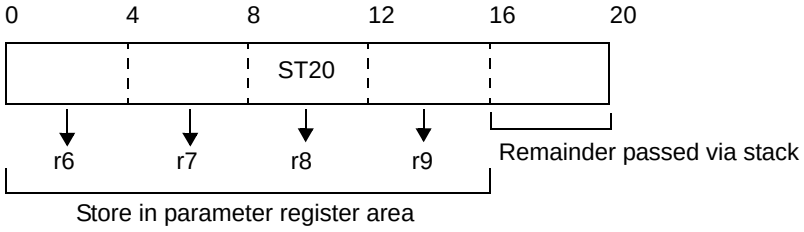
(2) Parameter register area

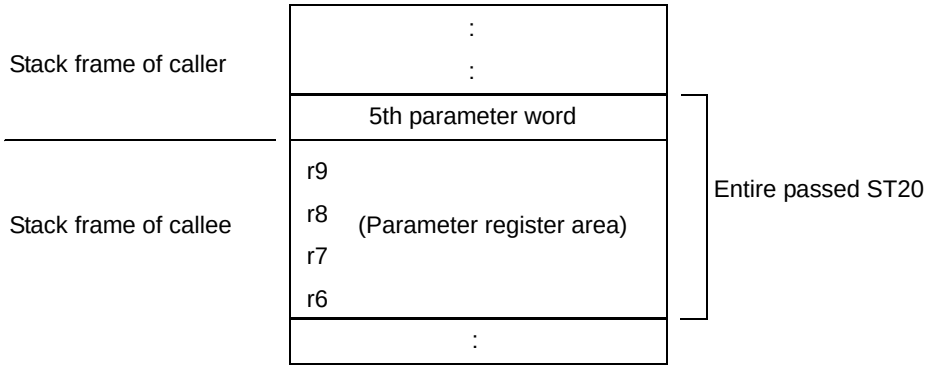
If the size of the parameters is greater than 4 words (16 bytes), then the required area for the size of the parameter register area is allocated. The size of this area will be either 0, 4, 8, 12, or 16 bytes, because it stores parameter registers r6 to r9, as necessary.
This area is for storing parameter 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.

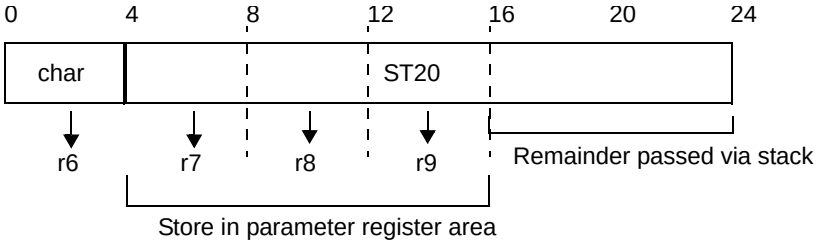




Below is a concrete case of parameters where this area is needed.
If none of these apply, then the parameter register area is not needed (size 0) because it is not necessary to store the parameter registers in the parameter register area.

(a) When a structure or union spans the parameter registers and stack

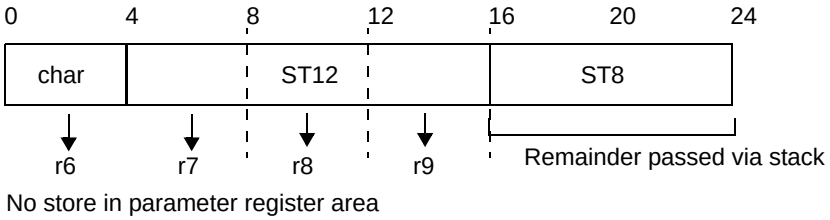
Example Function prototype : f(char, ST20)



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.

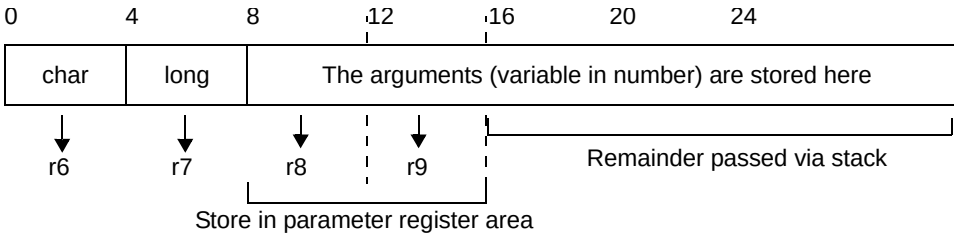
Example Function prototype : f(char, ST12, ST8)



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   SDATA
_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, fatal error message, abort 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
Message Numbers : 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	Fatal error : Processing is aborted if a set number of errors occur. No object codes are generated.
F	Abort error : Processing is aborted. No object codes are generated.
M	Information : Processing continues. Object codes are generated.
W	Warning : Processing continues. Object codes are generated (They might not be what the user intended).

10.4 Message Numbers

This section describes the message numbers displayed by CC-RH.
The message numbers when the CC-RH is executed are 5 digits number output following number (05).

10.5 Messages

This section describes the messages displayed by CC-RH.

10.5.1 Internal errors

Table 10.2 Internal Errors

C05*nnnnn*	[Message]	Internal error.
	[Message]	Internal error (*information*).
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0510000	[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.
C0560901	[Message]	Internal error. (Illegal linkage status (*number*))
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0560903	[Message]	Internal error. (Can not allocate memory(builtin new error)
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0560904	[Message]	Internal error. (No relocation type (*number*))
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0560905	[Message]	Internal error. (Illegal "*string*")
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0560906	[Message]	Internal error. (Extern undefined symbol exists ("*string*"))
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0560907	[Message]	Internal error. (Can not get the target dependent symbol:GP-symbol)
	[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.
C0592*nnn*	[Message]	Internal error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.

C0592100	[Message]	Internal error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.
C0592200	[Message]	Internal error.
	[Action by User]	Please contact your vendor or your Renesas Electronics overseas representative.

10.5.2 Fatal errors

Table 10.3 Fatal 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.
E0511111	[Message]	The "*character string*" option can not have a parameter.
E0511112	[Message]	The "*character string*" option requires a parameter.
	[Explanation]	The "*character string*" option requires a parameter. Specify the parameter.
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.
E0511118	[Message]	Symbol is required for the "*character string*" option.
E0511119	[Message]	The register "*register*" specified by the "-Xr"' option has been reserved by *compiler-name*.
E0511120	[Message]	Specify a value (*value1* - *value2*) for the "*character string*" option.
	[Explanation]	The value of the specified size option is outside the range of minimum value to maximum value.
	[Action by User]	Specify a size option value between the minimum and maximum values.
E0511121	[Message]	Multiple source files are not allowed when both the "-o" option and the "*character string*" option are specified.
E0511122	[Message]	The argument for the "*character string*" option must be an object file.
E0511124	[Message]	[CC-RH] The "-Xcommon" option must be specified.
	[Message]	[CC-RH] The "-cpu" option must be specified.
E0511125	[Message]	Cannot find device file.
E0511126	[Message]	Device file "*file*" read error.
E0511127	[Message]	The specified device is not supported.
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 "*character string*" option must be a folder when multiple source files are specified.
E0511134	[Message]	Input file "*file*" is not found.
E0511135	[Message]	"*path*" specified as an input file is a folder.
E0511136	[Message]	Failed to delete a temporary file "*file*".
E0511137	[Message]	Failed to delete a temporary folder "*folder*".
E0511138	[Message]	Failed to open an input file "*file*".
E0511139	[Message]	Failed to open an output file "*file*".
E0511140	[Message]	Failed to close an input file "*file*".
E0511141	[Message]	Failed to write an output file "*file*".
E0511142	[Message]	Multiple source files are not allowed when the "*character string*" option is specified.
E0511145	[Message]	"*character string2*" specified in the "*character string1*" option is not available.
E0511148	[Message]	"*file name*" is specified as an output file for the different options.
	[Action by User]	"*file name*" is specified as an output file for the different options. Specify a different file name.
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*.
E0511155	[Message]	The "*character string*" option needs other option(s).
E0511157	[Message]	The "*character string1*" option or the "*character string2*" option must be specified for this device.
E0511158	[Message]	The "*character string*" option is not supported for this device.
E0511159	[Message]	When the "*character string*" option is specified, source files cannot be input.
E0511160	[Message]	The "*character string*" option must be specified for this device.
E0511161	[Message]	Failed to delete a file "*file*".
E0511167	[Message]	Illegal section naming.
	[Explanation]	There is an error in section naming. The same section name is specified for different use of the section.
E0511173	[Message]	Failed to access a temporary file
E0511175	[Message]	Neither isa nor cpu is specified.
E0511176	[Message]	Both "-isa" option and "-cpu" option are specified.
E0511200	[Message]	Internal error(*error-information*).
E0512001	[Message]	Failed to delete a temporary file "*file*".
E0520001	[Message]	Last line of file ends without a newline.
	[Action by User]	The last line in the file does not end with a line break. Add a line break.

E0520002	[Message]	Last line of file ends with a backslash.
	[Explanation]	There is a backslash at the end of the last line of the file. Delete it.
E0520005	[Message]	Could not open source file "*file name*".
E0520006	[Message]	Comment unclosed at end of file.
	[Action by User]	There is an unclosed comment at the end of the file. Make sure that there are no unclosed comments.
E0520007	[Message]	Unrecognized token.
	[Action by User]	Unknown token. Check the indicated location.
E0520008	[Message]	Missing closing quote.
	[Action by User]	The string is missing a closing quotation mark. Make sure that there are no unclosed quotation mark.
E0520010	[Message]	"#" not expected here.
	[Explanation]	There is a "#" character in an invalid location.
E0520011	[Message]	Unrecognized preprocessing directive.
E0520012	[Message]	Parsing restarts here after previous syntax error.
E0520013	[Message]	Expected a file name.
E0520014	[Message]	Extra text after expected end of preprocessing directive.
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 "}".
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.
E0520096	[Message]	A translation unit must contain at least one declaration.
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.
E0520103	[Message]	Class is too large.
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.
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.
E0520123	[Message]	case label value has already appeared in this switch.
E0520124	[Message]	default label has already appeared in this switch.
E0520125	[Message]	Expected a "(".
E0520126	[Message]	Expression must be an lvalue.
E0520127	[Message]	Expected a statement.
E0520129	[Message]	A block-scope function may only have extern storage class.
E0520130	[Message]	Expected a "{".
E0520131	[Message]	Expression must have pointer-to-class type.
E0520132	[Message]	Expression must have pointer-to-struct-or-union type.
E0520133	[Message]	Expected a member name.
E0520134	[Message]	Expected a field name.
E0520135	[Message]	*symbol* has no member *member*.
E0520136	[Message]	*Type* "*symbol*" has no field "*field*".
E0520137	[Message]	Expression must be a modifiable value.
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.
E0520150	[Message]	A type name may not be redeclared as a parameter.
E0520151	[Message]	A typedef name may not be redeclared as a parameter.
E0520153	[Message]	Expression must have class type.

E0520154	[Message]	Expression must have struct or union type.
E0520157	[Message]	Expression must be an integral constant expression.
E0520158	[Message]	Expression must be an lvalue or a function designator.
E0520159	[Message]	Declaration is incompatible with previous "*declaration*" (declared at line *number*).
E0520160	[Message]	External name conflicts with external name of "*symbol*".
E0520165	[Message]	Too few arguments in function call.
E0520166	[Message]	Invalid floating constant.
E0520167	[Message]	Argument of type "*type1*" is incompatible with parameter of type "*type2*".
E0520168	[Message]	A function type is not allowed here.
E0520169	[Message]	Expected a declaration.
E0520171	[Message]	Invalid type conversion.
E0520172	[Message]	External/internal linkage conflict with previous declaration.
E0520173	[Message]	Floating-point value does not fit in required integral type.
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.
E0520194	[Message]	Expected an asm string.
	[Explanation]	There is no assembler string in an "__asm()" statement.
E0520195	[Message]	An asm function must be prototyped.
E0520196	[Message]	An asm function may not have an ellipsis
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.
E0520227	[Message]	Macro recursion.
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.
E0520239	[Message]	Invalid specifier outside a class declaration.
E0520240	[Message]	Duplicate specifier in declaration.
E0520241	[Message]	A union is not allowed to have a base class.
E0520242	[Message]	Multiple access control specifiers are not allowed.
E0520243	[Message]	class or struct definition is missing.
E0520244	[Message]	Qualified name is not a member of class *type* or its base classes.
E0520245	[Message]	A nonstatic member reference must be relative to a specific object.

E0520246	[Message]	A nonstatic data member may not be defined outside its class.
E0520247	[Message]	*Type* "*symbol*" has already been defined.
E0520248	[Message]	Pointer to reference is not allowed.
E0520249	[Message]	Reference to reference is not allowed.
E0520250	[Message]	Reference to void is not allowed.
E0520251	[Message]	Array of reference is not allowed.
E0520252	[Message]	Reference "*name*" requires an initializer.
E0520253	[Message]	Expected a ",".
E0520254	[Message]	Type name is not allowed.
E0520255	[Message]	Type definition is not allowed.
E0520256	[Message]	Invalid redeclaration of type name "*type*".
	[Explanation]	Type name "*type*" was redeclared illegally.
E0520257	[Message]	const *type* "*symbol*" requires an initializer.
E0520258	[Message]	"this" may only be used inside a nonstatic member function
E0520259	[Message]	Constant value is not known.
E0520260	[Message]	Explicit type is missing ("int" assumed).
E0520262	[Message]	Not a class or struct name.
E0520263	[Message]	Duplicate base class name.
E0520264	[Message]	Invalid base class.
E0520265	[Message]	"*name*" is inaccessible.
E0520266	[Message]	"*name*" is ambiguous.
E0520267	[Message]	Old-style parameter list (anachronism).
E0520268	[Message]	Declaration may not appear after executable statement in block.
E0520269	[Message]	Conversion to inaccessible base class "*type*" is not allowed.
E0520274	[Message]	Improperly terminated macro invocation.
E0520276	[Message]	Name followed by "::" must be a class or namespace name.
E0520277	[Message]	Invalid friend declaration.
E0520278	[Message]	A constructor or destructor may not return a value.
E0520279	[Message]	Invalid destructor declaration.
E0520280	[Message]	Declaration of a member with the same name as its class.
E0520281	[Message]	Global-scope qualifier (leading "::") is not allowed.
E0520282	[Message]	The global scope has no *xxx*.
E0520283	[Message]	Qualified name is not allowed.
E0520284	[Message]	NULL reference is not allowed.
E0520285	[Message]	Initialization with "{...}" is not allowed for object of type "*type*".
E0520286	[Message]	Base class "*type*" is ambiguous.

E0520287	[Message]	Derived class *type1* contains more than one instance of class *type2*.
E0520288	[Message]	Cannot convert pointer to base class *type2* to pointer to derived class *type1* -- base class is virtual.
E0520289	[Message]	No instance of constructor *name* matches the argument list.
E0520290	[Message]	Copy constructor for class *type* is ambiguous.
E0520291	[Message]	No default constructor exists for class *type*.
E0520292	[Message]	*name* is not a nonstatic data member or base class of class *type*.
E0520293	[Message]	Indirect nonvirtual base class is not allowed.
E0520294	[Message]	Invalid union member -- class *type* has a disallowed member function.
E0520296	[Message]	Invalid use of non-lvalue array.
E0520297	[Message]	Expected an operator.
E0520298	[Message]	Inherited member is not allowed.
E0520299	[Message]	Cannot determine which instance of *name* is intended.
E0520300	[Message]	A pointer to a bound function may only be used to call the function.
E0520301	[Message]	typedef name has already been declared (with same type).
E0520302	[Message]	Symbol has already been defined.
E0520304	[Message]	No instance of *name* matches the argument list.
E0520305	[Message]	Type definition is not allowed in function return type declaration.
E0520306	[Message]	Default argument not at end of parameter list.
E0520307	[Message]	Redefinition of default argument.
E0520308	[Message]	More than one instance of *name* matches the argument list:
E0520309	[Message]	More than one instance of constructor *name* matches the argument list:
E0520310	[Message]	Default argument of type *type1* is incompatible with parameter of type *type2*.
E0520311	[Message]	Cannot overload functions distinguished by return type alone.
E0520312	[Message]	No suitable user-defined conversion from *type1* to *type2* exists.
E0520313	[Message]	Type qualifier is not allowed on this function.
E0520314	[Message]	Only nonstatic member functions may be virtual.
E0520315	[Message]	The object has cv-qualifiers that are not compatible with the member function.
E0520316	[Message]	Program too large to compile (too many virtual functions).
E0520317	[Message]	Return type is not identical to nor covariant with return type *type* of overridden virtual function *name*.
E0520318	[Message]	Override of virtual *name* is ambiguous.
E0520319	[Message]	Pure specifier ("= 0") allowed only on virtual functions.
E0520320	[Message]	Badly-formed pure specifier (only "= 0" is allowed).
E0520321	[Message]	Data member initializer is not allowed.
E0520322	[Message]	Object of abstract class type *type* is not allowed:
E0520323	[Message]	function returning abstract class *type* is not allowed:

E0520325	[Message]	inline specifier allowed on function declarations only.
E0520326	[Message]	inline is not allowed.
E0520327	[Message]	Invalid storage class for an inline function.
E0520328	[Message]	Invalid storage class for a class member.
E0520329	[Message]	Local class member *name* requires a definition.
E0520330	[Message]	*name* is inaccessible.
E0520332	[Message]	class *type* has no copy constructor to copy a const object.
E0520333	[Message]	Defining an implicitly declared member function is not allowed.
E0520334	[Message]	class *type* has no suitable copy constructor.
E0520335	[Message]	Linkage specification is not allowed.
E0520336	[Message]	Unknown external linkage specification.
E0520337	[Message]	Linkage specification is incompatible with previous "*symbol*".
E0520338	[Message]	More than one instance of overloaded function *name* has "C" linkage.
E0520339	[Message]	class *type* has more than one default constructor.
E0520340	[Message]	Value copied to temporary, reference to temporary used.
E0520341	[Message]	"operator *operator*" must be a member function.
E0520342	[Message]	Operator may not be a static member function.
E0520343	[Message]	No arguments allowed on user-defined conversion.
E0520344	[Message]	Too many parameters for this operator function.
E0520345	[Message]	Too few parameters for this operator function.
E0520346	[Message]	Nonmember operator requires a parameter with class type.
E0520347	[Message]	Default argument is not allowed.
E0520348	[Message]	More than one user-defined conversion from *type1* to *type2* applies:
E0520349	[Message]	No operator *operator* matches these operands.
E0520350	[Message]	More than one operator *operator* matches these operands:
E0520351	[Message]	First parameter of allocation function must be of type "size_t".
E0520352	[Message]	Allocation function requires "void *" return type.
E0520353	[Message]	Deallocation function requires "void" return type.
E0520354	[Message]	First parameter of deallocation function must be of type "void *".
E0520356	[Message]	Type must be an object type.
E0520357	[Message]	Base class *xxx* has already been initialized.
E0520358	[Message]	Base class name required -- *xxx* assumed (anachronism).
E0520359	[Message]	Symbol has already been initialized.
E0520360	[Message]	Name of member or base class is missing.
E0520363	[Message]	Invalid anonymous union -- nonpublic member is not allowed.
E0520364	[Message]	Invalid anonymous union -- member function is not allowed.

E0520365	[Message]	Anonymous union at global or namespace scope must be declared static.
E0520366	[Message]	Symbol provides no initializer for:
E0520367	[Message]	Implicitly generated constructor for class *type* cannot initialize:
E0520369	[Message]	*name* has an uninitialized const or reference member.
E0520371	[Message]	class *type* has no assignment operator to copy a const object.
E0520372	[Message]	class *type* has no suitable assignment operator.
E0520373	[Message]	Ambiguous assignment operator for class*type*.
E0520375	[Message]	Declaration requires a typedef name.
E0520378	[Message]	static is not allowed.
E0520380	[Message]	Expression must have pointer-to-member type.
E0520384	[Message]	No instance of overloaded *name* matches the argument list.
E0520386	[Message]	No instance of *name* matches the required type.
E0520389	[Message]	A cast to abstract class *type* is not allowed:
E0520390	[Message]	Function "main" may not be called or have its address taken.
E0520391	[Message]	A new-initializer may not be specified for an array.
E0520392	[Message]	Member function *name* may not be redeclared outside its class.
E0520393	[Message]	Pointer to incomplete class type is not allowed.
E0520394	[Message]	Reference to local variable of enclosing function is not allowed.
E0520397	[Message]	Implicitly generated assignment operator cannot copy:
E0520401	[Message]	Destructor for base class *type* is not virtual.
E0520403	[Message]	Invalid redeclaration of member "symbol".
E0520404	[Message]	Function "main" may not be declared inline.
E0520405	[Message]	Member function with the same name as its class must be a constructor.
E0520407	[Message]	A destructor may not have parameters.
E0520408	[Message]	Copy constructor for class *type1* may not have a parameter of type *type2*.
E0520409	[Message]	*Type* "*symbol*" returns incomplete type "*type*".
E0520410	[Message]	Protected *name* is not accessible through a *type* pointer or object.
E0520411	[Message]	A parameter is not allowed.
E0520412	[Message]	An "__asm" declaration is not allowed here.
E0520413	[Message]	No suitable conversion function from *type1* to *type2* exists.
E0520415	[Message]	No suitable constructor exists to convert from *type1* to *type2*.
E0520416	[Message]	More than one constructor applies to convert from *type1* to *type2*:
E0520417	[Message]	More than one conversion function from *type1* to *type2* applies:
E0520418	[Message]	More than one conversion function from *type* to a built-in type applies:
E0520424	[Message]	A constructor or destructor may not have its address taken.
E0520427	[Message]	Qualified name is not allowed in member declaration.

E0520429	[Message]	The size of an array in "new" must be non-negative.
E0520432	[Message]	enum declaration is not allowed.
E0520433	[Message]	Qualifiers dropped in binding reference of type *type1* to initializer of type *type2*.
E0520434	[Message]	A reference of type *type1* (not const-qualified) cannot be initialized with a value of type *type2.*
E0520435	[Message]	A pointer to function may not be deleted.
E0520436	[Message]	Conversion function must be a nonstatic member function.
E0520437	[Message]	Template declaration is not allowed here.
E0520438	[Message]	Expected a "<".
E0520439	[Message]	Expected a ">".
E0520440	[Message]	Template parameter declaration is missing.
E0520441	[Message]	Argument list for "*name*" is missing.
E0520442	[Message]	Too few arguments for "*name*".
E0520443	[Message]	Too many arguments for "symbol".
E0520445	[Message]	*name1* is not used in declaring the parameter types of "*name2*".
E0520449	[Message]	More than one instance of *name* matches the required type.
E0520450	[Message]	The type "long long" is nonstandard.
E0520451	[Message]	Omission of "class" is nonstandard.
E0520452	[Message]	Return type may not be specified on a conversion function.
E0520456	[Message]	Excessive recursion at instantiation of *name*.
E0520457	[Message]	*name* is not a function or static data member.
E0520458	[Message]	Argument of type *type1* is incompatible with template parameter of type *type2*.
E0520459	[Message]	Initialization requiring a temporary or conversion is not allowed.
E0520460	[Message]	declaration of *xxx* hides function parameter.
E0520461	[Message]	Initial value of reference to non-const must be an lvalue.
E0520463	[Message]	"template" is not allowed.
E0520464	[Message]	*type* is not a class template.
E0520466	[Message]	"main" is not a valid name for a function template.
E0520467	[Message]	Invalid reference to *name* (union/nonunion mismatch).
E0520468	[Message]	A template argument may not reference a local type.
E0520469	[Message]	Tag kind of *xxx* is incompatible with declaration of "symbol".
E0520470	[Message]	The global scope has no tag named *xxx*.
E0520471	[Message]	symbol has no tag member named *xxx*.
E0520473	[Message]	*name* may be used only in pointer-to-member declaration.
E0520475	[Message]	A template argument may not reference a non-external entity.
E0520476	[Message]	Name followed by "::~" must be a class name or a type name.
E0520477	[Message]	Destructor name does not match name of class *type*.

E0520478	[Message]	Type used as destructor name does not match type *type*.
E0520481	[Message]	Invalid storage class for a template declaration.
E0520484	[Message]	Invalid explicit instantiation declaration.
E0520485	[Message]	*name* is not an entity that can be instantiated.
E0520486	[Message]	Compiler generated *name* cannot be explicitly instantiated.
E0520487	[Message]	Inline *name* cannot be explicitly instantiated.
E0520489	[Message]	*name* cannot be instantiated -- no template definition was supplied.
E0520490	[Message]	*name* cannot be instantiated -- it has been explicitly specialized.
E0520493	[Message]	No instance of *name* matches the specified type.
E0520494	[Message]	Declaring a void parameter list with a typedef is nonstandard.
E0520496	[Message]	Template parameter *name* may not be redeclared in this scope.
E0520498	[Message]	Template argument list must match the parameter list.
E0520500	[Message]	Extra parameter of postfix "operator *xxx*" must be of type "int".
E0520501	[Message]	An operator name must be declared as a function.
E0520502	[Message]	Operator name is not allowed.
E0520503	[Message]	*name* cannot be specialized in the current scope.
E0520504	[Message]	Nonstandard form for taking the address of a member function.
E0520505	[Message]	Too few template parameters -- does not match previous declaration.
E0520506	[Message]	Too many template parameters -- does not match previous declaration.
E0520507	[Message]	Function template for operator delete(void *) is not allowed.
E0520508	[Message]	class template and template parameter may not have the same name.
E0520510	[Message]	A template argument may not reference an unnamed type.
E0520511	[Message]	Enumerated type is not allowed.
E0520513	[Message]	A value of type "type1" cannot be assigned to an entity of type "type2".
E0520515	[Message]	Cannot convert to incomplete class *type*.
E0520516	[Message]	const object requires an initializer.
E0520517	[Message]	Object has an uninitialized const or reference member.
E0520518	[Message]	Nonstandard preprocessing directive.
E0520519	[Message]	*name* may not have a template argument list.
E0520520	[Message]	Initialization with "{...}" expected for aggregate object.
E0520521	[Message]	Pointer-to-member selection class types are incompatible (*type1* and *type2*).
E0520525	[Message]	A dependent statement may not be a declaration.
	[Explanation]	Cannot write declaration due to lack of "{" character after "if()" statement.
E0520526	[Message]	A parameter may not have void type.
E0520529	[Message]	This operator is not allowed in a template argument expression.
E0520530	[Message]	Try block requires at least one handler/

E0520531	[Message]	Handler requires an exception declaration.
E0520532	[Message]	Handler is masked by default handler.
E0520536	[Message]	Exception specification is incompatible with that of previous *name*.
E0520540	[Message]	Support for exception handling is disabled.
E0520543	[Message]	Non-arithmetic operation not allowed in nontype template argument.
E0520544	[Message]	Use of a local type to declare a nonlocal variable.
E0520545	[Message]	Use of a local type to declare a function.
E0520546	[Message]	Transfer of control bypasses initialization of:
E0520548	[Message]	Transfer of control into an exception handler.
E0520551	[Message]	symbol cannot be defined in the current scope.
E0520555	[Message]	Tag kind of *name* is incompatible with template parameter of type *type*.
E0520556	[Message]	Function template for operator new(size_t) is not allowed.
E0520558	[Message]	Pointer to member of type "*type*" is not allowed.
E0520559	[Message]	Tointer to member of type *type* is not allowed.
E0520560	[Message]	symbol is reserved for future use as a keyword.
E0520561	[Message]	Invalid macro definition:
E0520562	[Message]	Invalid macro undefinition:
E0520598	[Message]	A template parameter may not have void type.
E0520599	[Message]	Excessive recursive instantiation of *name* due to instantiate-all mode.
E0520601	[Message]	A throw expression may not have void type.
E0520603	[Message]	Parameter of abstract class type *type* is not allowed:
E0520604	[Message]	Array of abstract class *type* is not allowed:
E0520605	[Message]	Floating-point template parameter is nonstandard.
E0520606	[Message]	This pragma must immediately precede a declaration.
E0520607	[Message]	This pragma must immediately precede a statement.
E0520608	[Message]	This pragma must immediately precede a declaration or statement.
E0520609	[Message]	This kind of pragma may not be used here.
E0520612	[Message]	Specific definition of inline template function must precede its first use.
E0520615	[Message]	Parameter type involves pointer to array of unknown bound.
E0520616	[Message]	Parameter type involves reference to array of unknown bound.
E0520618	[Message]	struct or union declares no named members.
E0520619	[Message]	Nonstandard unnamed field.
E0520620	[Message]	Nonstandard unnamed member.
E0520643	[Message]	restrict is not allowed.
E0520644	[Message]	A pointer or reference to function type may not be qualified by "restrict".
E0520647	[Message]	Conflicting calling convention modifiers.

E0520651	[Message]	A calling convention may not be followed by a nested declarator.
E0520654	[Message]	Declaration modifiers are incompatible with previous declaration.
E0520656	[Message]	Transfer of control into a try block.
E0520658	[Message]	Closing brace of template definition not found.
E0520660	[Message]	Invalid packing alignment value.
E0520661	[Message]	Expected an integer constant.
E0520663	[Message]	Invalid source file identifier string.
E0520664	[Message]	A class template cannot be defined in a friend declaration.
E0520665	[Message]	asm is not allowed.
E0520666	[Message]	asm must be used with a function definition.
E0520667	[Message]	asm function is nonstandard.
E0520668	[Message]	Ellipsis with no explicit parameters is nonstandard.
E0520669	[Message]	&... is nonstandard.
E0520670	[Message]	invalid use of "&...".
E0520673	[Message]	A reference of type *type1* cannot be initialized with a value of type *type2*.
E0520674	[Message]	Initial value of reference to const volatile must be an lvalue.
E0520676	[Message]	Using out-of-scope declaration of *type* "*symbol*" (declared at line *number*).
E0520691	[Message]	*xxx*, required for copy that was eliminated, is inaccessible.
E0520692	[Message]	*xxx* required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
E0520693	[Message]	<typeinfo> must be included before typeid is used.
E0520694	[Message]	*xxx* cannot cast away const or other type qualifiers.
E0520695	[Message]	The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *.
E0520696	[Message]	The operand of a pointer dynamic_cast must be a pointer to a complete class type.
E0520697	[Message]	The operand of a reference dynamic_cast must be an lvalue of a complete class type.
E0520698	[Message]	The operand of a runtime dynamic_cast must have a polymorphic class type.
E0520701	[Message]	An array type is not allowed here.
E0520702	[Message]	Expected an "=".
E0520703	[Message]	Expected a declarator in condition declaration.
E0520704	[Message]	*xxx*, declared in condition, may not be redeclared in this scope.
E0520705	[Message]	Default template arguments are not allowed for function templates.
E0520706	[Message]	Expected a "," or ">".
E0520707	[Message]	Expected a template parameter list.
E0520709	[Message]	bool type is not allowed.
E0520710	[Message]	Offset of base class *name1* within class *name2* is too large.

E0520711	[Message]	Expression must have bool type (or be convertible to bool).
E0520717	[Message]	The type in a const_cast must be a pointer, reference, or pointer to member to an object type.
E0520718	[Message]	A const_cast can only adjust type qualifiers; it cannot change the underlying type.
E0520719	[Message]	mutable is not allowed.
E0520724	[Message]	namespace definition is not allowed.
E0520725	[Message]	name must be a namespace name.
E0520726	[Message]	namespace alias definition is not allowed.
E0520727	[Message]	namespace-qualified name is required.
E0520728	[Message]	A namespace name is not allowed.
E0520730	[Message]	*name* is not a class template.
E0520731	[Message]	Array with incomplete element type is nonstandard.
E0520732	[Message]	Allocation operator may not be declared in a namespace.
E0520733	[Message]	Deallocation operator may not be declared in a namespace.
E0520734	[Message]	*name1* conflicts with using-declaration of *name2*.
E0520735	[Message]	using-declaration of *name1* conflicts with *name2*.
E0520742	[Message]	symbol has no actual member *xxx*.
E0520749	[Message]	A type qualifier is not allowed.
E0520750	[Message]	*name* was used before its template was declared.
E0520751	[Message]	Static and nonstatic member functions with same parameter types cannot be overloaded.
E0520752	[Message]	No prior declaration of "symbol".
E0520753	[Message]	A template-id is not allowed.
	[Explanation]	The use of templates (*template name*<template *argument*>) is not allowed.
E0520754	[Message]	A class-qualified name is not allowed.
E0520755	[Message]	symbol may not be redeclared in the current scope.
E0520756	[Message]	Qualified name is not allowed in namespace member declaration.
E0520757	[Message]	symbol is not a type name.
E0520758	[Message]	Explicit instantiation is not allowed in the current scope.
E0520759	[Message]	*symbol* cannot be explicitly instantiated in the current scope.
E0520761	[Message]	typename may only be used within a template.
E0520765	[Message]	Nonstandard character at start of object-like macro definition.
E0520766	[Message]	Exception specification for virtual *name1* is incompatible with that of overridden *name2*.
E0520767	[Message]	Conversion from pointer to smaller integer.
E0520768	[Message]	Exception specification for implicitly declared virtual *name1* is incompatible with that of overridden *name2*.
E0520769	[Message]	*name1*, implicitly called from *name2*, is ambiguous.

E0520771	[Message]	"explicit" is not allowed.
E0520772	[Message]	Declaration conflicts with *xxx* (reserved class name).
E0520773	[Message]	Only "()" is allowed as initializer for array "symbol".
E0520774	[Message]	"virtual" is not allowed in a function template declaration.
E0520775	[Message]	Invalid anonymous union -- class member template is not allowed.
E0520776	[Message]	Template nesting depth does not match the previous declaration of %n.
E0520779	[Message]	*xxx*, declared in for-loop initialization, may not be redeclared in this scope.
E0520782	[Message]	Definition of virtual *name* is required here.
E0520784	[Message]	A storage class is not allowed in a friend declaration.
E0520785	[Message]	Template parameter list for *name* is not allowed in this declaration.
E0520786	[Message]	*name* is not a valid member class or function template.
E0520787	[Message]	Not a valid member class or function template declaration.
E0520788	[Message]	A template declaration containing a template parameter list may not be followed by an explicit specialization declaration.
E0520789	[Message]	Explicit specialization of *name1* must precede the first use of *name2*.
E0520790	[Message]	Explicit specialization is not allowed in the current scope.
E0520791	[Message]	Partial specialization of "symbol" is not allowed.
E0520792	[Message]	*name* is not an entity that can be explicitly specialized.
E0520793	[Message]	Explicit specialization of %n must precede its first use.
E0520795	[Message]	Specializing *name* requires "template<>" syntax.
E0520799	[Message]	Specializing *symbol* without "template<>" syntax is nonstandard.
E0520800	[Message]	This declaration may not have extern "C" linkage.
E0520801	[Message]	*name* is not a class or function template name in the current scope.
E0520803	[Message]	Specifying a default argument when redeclaring an already referenced function template is not allowed.
E0520804	[Message]	Cannot convert pointer to member of base class type*2* to pointer to member of derived class type*1* -- base class is virtual.
E0520805	[Message]	Exception specification is incompatible with that of *name*.
E0520807	[Message]	Unexpected end of default argument expression.
E0520808	[Message]	Default-initialization of reference is not allowed.
E0520809	[Message]	Uninitialized "symbol" has a const member.
E0520810	[Message]	Uninitialized base class type has a const member.
E0520811	[Message]	const *name* requires an initializer -- class *type* has no explicitly declared default constructor.
E0520812	[Message]	Const object requires an initializer -- class *type* has no explicitly declared default constructor.
E0520816	[Message]	In a function definition a type qualifier on a "void" return type is not allowed.
E0520817	[Message]	Static data member declaration is not allowed in this class.

E0520818	[Message]	Template instantiation resulted in an invalid function declaration.
E0520819	[Message]	... is not allowed.
E0520822	[Message]	Invalid destructor name for type *type*.
E0520824	[Message]	Destructor reference is ambiguous -- both *name1* and *name2* could be used.
E0520827	[Message]	Only one member of a union may be specified in a constructor initializer list.
E0520828	[Message]	Support for "new[]" and "delete[]" is disabled.
E0520832	[Message]	No appropriate operator delete is visible.
E0520833	[Message]	Pointer or reference to incomplete type is not allowed.
E0520834	[Message]	Invalid partial specialization -- *name* is already fully specialized.
E0520835	[Message]	Incompatible exception specifications.
E0520840	[Message]	A template argument list is not allowed in a declaration of a primary template.
E0520841	[Message]	Partial specializations may not have default template arguments.
E0520842	[Message]	*name1* is not used in or cannot be deduced from the template argument list of *name2*.
E0520844	[Message]	The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter.
E0520845	[Message]	This partial specialization would have been used to instantiate *name*.
E0520846	[Message]	This partial specialization would have made the instantiation of *name* ambiguous.
E0520847	[Message]	Expression must have integral or enum type.
E0520848	[Message]	Expression must have arithmetic or enum type.
E0520849	[Message]	Expression must have arithmetic, enum, or pointer type.
E0520850	[Message]	Type of cast must be integral or enum.
E0520851	[Message]	Type of cast must be arithmetic, enum, or pointer.
E0520852	[Message]	Expression must be a pointer to a complete object type.
E0520854	[Message]	A partial specialization nontype argument must be the name of a nontype parameter or a constant.
E0520855	[Message]	Return type is not identical to return type *type* of overridden virtual function *name*.
E0520857	[Message]	A partial specialization of a class template must be declared in the namespace of which it is a member.
E0520858	[Message]	*name* is a pure virtual function.
E0520859	[Message]	Pure virtual *name* has no overrider.
E0520861	[Message]	Invalid character in input line.
E0520862	[Message]	Function returns incomplete type "*type*".
E0520864	[Message]	*name* is not a template.
E0520865	[Message]	A friend declaration may not declare a partial specialization.
E0520868	[Message]	Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator).
E0520870	[Message]	Invalid multibyte character sequence.

E0520871	[Message]	Template instantiation resulted in unexpected function type of *type1* (the meaning of a name may have changed since the template declaration -- the type of the template is *type2*).
E0520872	[Message]	Ambiguous guiding declaration -- more than one function template *name* matches type *type*.
E0520873	[Message]	Non-integral operation not allowed in nontype template argument.
E0520875	[Message]	Embedded C++ does not support templates.
E0520876	[Message]	Embedded C++ does not support exception handling.
E0520877	[Message]	Embedded C++ does not support namespaces.
E0520878	[Message]	Embedded C++ does not support run-time type information.
E0520879	[Message]	Embedded C++ does not support the new cast syntax.
E0520880	[Message]	Embedded C++ does not support using-declarations.
E0520881	[Message]	Embedded C++ does not support \"mutable\".
E0520882	[Message]	Embedded C++ does not support multiple or virtual inheritance.
E0520885	[Message]	*type1* cannot be used to designate constructor for *type2*.
E0520886	[Message]	Invalid suffix on integral constant.
	[Explanation]	The integer constant has an invalid suffix.
E0520890	[Message]	Variable length array with unspecified bound is not allowed.
E0520891	[Message]	An explicit template argument list is not allowed on this declaration.
E0520892	[Message]	An entity with linkage cannot have a type involving a variable length array.
E0520893	[Message]	A variable length array cannot have static storage duration.
E0520894	[Message]	Entity-kind "*name*" is not a template.
E0520896	[Message]	Expected a template argument.
E0520898	[Message]	Nonmember operator requires a parameter with class or enum type.
E0520901	[Message]	Qualifier of destructor name *type1* does not match type *type2*.
E0520915	[Message]	A segment name has already been specified.
E0520916	[Message]	Cannot convert pointer to member of derived class *type1* to pointer to member of base class *type2* -- base class is virtual.
E0520928	[Message]	Incorrect use of va_start.
E0520929	[Message]	Incorrect use of va_arg.
E0520930	[Message]	Incorrect use of va_end.
E0520934	[Message]	A member with reference type is not allowed in a union.
E0520935	[Message]	Typedef may not be specified here.
E0520937	[Message]	A class or namespace qualified name is required.
E0520938	[Message]	Return type "int" omitted in declaration of function "main".
E0520939	[Message]	Pointer-to-member representation *xxx* is too restrictive for *xxx*.
E0520940	[Message]	Missing return statement at end of non-void *type* "*symbol*".
E0520946	[Message]	Name following "template" must be a template.

E0520948	[Message]	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
E0520951	[Message]	Return type of function "main" must be "int".
E0520952	[Message]	A nontype template parameter may not have class type.
E0520953	[Message]	A default template argument cannot be specified on the declaration of a member of a class template outside of its class.
E0520954	[Message]	A return statement is not allowed in a handler of a function try block of a constructor.
E0520955	[Message]	Ordinary and extended designators cannot be combined in an initializer designation.
E0520956	[Message]	The second subscript must not be smaller than the first.
E0520960	[Message]	Type used as constructor name does not match type *type*.
E0520961	[Message]	Use of a type with no linkage to declare a variable with linkage.
E0520962	[Message]	Use of a type with no linkage to declare a function.
E0520963	[Message]	Return type may not be specified on a constructor.
E0520964	[Message]	Return type may not be specified on a destructor.
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.
E0520971	[Message]	Array range designators cannot be applied to dynamic initializers.
E0520972	[Message]	Property name cannot appear here.
E0520975	[Message]	A variable-length array type is not allowed.
E0520976	[Message]	A compound literal is not allowed in an integral constant expression.
E0520977	[Message]	A compound literal of type "type" is not allowed.
E0520978	[Message]	A template friend declaration cannot be declared in a local class.
E0520979	[Message]	Ambiguous "?" operation: second operand of type *type1* can be converted to third operand type *type2*, and vice versa.
E0520980	[Message]	Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type.
E0520982	[Message]	There is more than one way an object of type "type" can be called for the argument list:
E0520983	[Message]	typedef name has already been declared (with similar type).
E0520985	[Message]	Storage class "mutable" is not allowed for anonymous unions.
E0520987	[Message]	Abstract class type *type* is not allowed as catch type:
E0520988	[Message]	A qualified function type cannot be used to declare a nonmember function or a static member function.

E0520989	[Message]	A qualified function type cannot be used to declare a parameter.
E0520990	[Message]	Cannot create a pointer or reference to qualified function type.
E0520992	[Message]	Invalid macro definition:.
E0520993	[Message]	Subtraction of pointer types "*type1*" and "*type2*" is nonstandard.
E0520994	[Message]	An empty template parameter list is not allowed in a template template parameter declaration.
E0520995	[Message]	Expected "class".
E0520996	[Message]	The "class" keyword must be used when declaring a template template parameter.
E0520998	[Message]	A qualified name is not allowed for a friend declaration that is a function definition.
E0520999	[Message]	*symbol1* is not compatible with "*symbol2*".
E0521001	[Message]	Class member designated by a using-declaration must be visible in a direct base class.
E0521006	[Message]	A template template parameter cannot have the same name as one of its template parameters.
E0521007	[Message]	Recursive instantiation of default argument.
E0521009	[Message]	*symbol* is not an entity that can be defined.
E0521010	[Message]	Destructor name must be qualified.
E0521011	[Message]	Friend class name may not be introduced with "type*name*".
E0521012	[Message]	A using-declaration may not name a constructor or destructor.
E0521013	[Message]	A qualified friend template declaration must refer to a specific previously declared template.
E0521014	[Message]	Invalid specifier in class template declaration.
E0521015	[Message]	Argument is incompatible with formal parameter.
E0521017	[Message]	Loop in sequence of "operator->" functions starting at class *xxx*.
E0521018	[Message]	*xxx* has no member class *xxx*.
E0521019	[Message]	The global scope has no class named *xxx*.
E0521020	[Message]	Recursive instantiation of template default argument.
E0521021	[Message]	Access declarations and using-declarations cannot appear in unions.
E0521022	[Message]	*xxx* is not a class member.
E0521023	[Message]	Nonstandard member constant declaration 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.
E0521032	[Message]	Argument type %t does not match this type-generic function macro.
E0521034	[Message]	Friend declaration cannot add default arguments to previous declaration.
E0521035	[Message]	*xxx* cannot be declared in this scope.
E0521036	[Message]	The reserved identifier "*symbol*" may only be used inside a function.

E0521037	[Message]	This universal character cannot begin an identifierl.
E0521038	[Message]	Expected a string literal.
E0521039	[Message]	Unrecognized STDC pragma.
E0521040	[Message]	Expected "ON", "OFF", or "DEFAULT".
E0521041	[Message]	A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.
E0521042	[Message]	Incorrect use of va_copy.
E0521043	[Message]	*xxx* can only be used with floating-point types.
E0521044	[Message]	Complex type is not allowed.
E0521045	[Message]	Invalid designator kind.
E0521047	[Message]	Complex floating-point operation result is out of range.
E0521048	[Message]	Conversion between real and imaginary yields zero.
E0521049	[Message]	An initializer cannot be specified for a flexible array member.
E0521051	[Message]	Standard requires that "symbol" be given a type by a subsequent declaration ("int" assumed).
E0521052	[Message]	A definition is required for inline "symbol".
E0521054	[Message]	A floating-point type must be included in the type specifier for a _Complex or _Imaginary type.
E0521055	[Message]	Types cannot be declared in anonymous unions.
E0521056	[Message]	Returning pointer to local variable.
E0521057	[Message]	Returning pointer to local temporary.
E0521061	[Message]	Declaration of "symbol" is incompatible with a declaration in another translation unit.
E0521062	[Message]	The other declaration is %p.
E0521065	[Message]	A field declaration cannot have a type involving a variable length array.
E0521066	[Message]	Declaration of "*symbol*" had a different meaning during compilation of *file*.
E0521067	[Message]	Eexpected "template".
E0521072	[Message]	A declaration cannot have a label.
E0521075	[Message]	"*symbol*" already defined during compilation of *xxx*.
E0521076	[Message]	"*symbol*" already defined in another translation unit.
E0521081	[Message]	A field with the same name as its class cannot be declared in a class with a user-declared constructor.
E0521086	[Message]	The object has cv-qualifiers that are not compatible with the member "*symbol*".
E0521087	[Message]	No instance of *xxx* matches the argument list and object (the object has cv-qualifiers that prevent a match).
E0521088	[Message]	An attribute specifies a mode incompatible with *xxx*.
E0521089	[Message]	There is no type with the width specified.
E0521139	[Message]	The "template" keyword used for syntactic disambiguation may only be used within a template.

E0521144	[Message]	Storage class must be auto or register.
E0521146	[Message]	*xxx* is not a base class member.
E0521158	[Message]	void return type cannot be qualified.
E0521161	[Message]	A member template corresponding to *xxx* is declared as a template of a different kind in another translation unit.
E0521163	[Message]	va_start should only appear in a function with an ellipsis parameter.
E0521201	[Message]	typedef *xxx* may not be used in an elaborated type specifier.
E0521203	[Message]	Parameter *parameter* may not be redeclared in a catch clause of function try block.
E0521204	[Message]	The initial explicit specialization of *xxx* must be declared in the namespace containing the template.
E0521206	[Message]	"template" must be followed by an identifier.
E0521212	[Message]	This pragma cannot be used in a _Pragma operator (a #pragma directive must be used).
E0521227	[Message]	Transfer of control into a statement expression is not allowed.
E0521229	[Message]	This statement is not allowed inside of a statement expression.
E0521230	[Message]	Anon-POD class definition is not allowed inside of a statement expression.
E0521254	[Message]	Integer overflow in internal computation due to size or complexity of "*type*".
E0521255	[Message]	Integer overflow in internal computation.
E0521273	[Message]	Alignment-of operator applied to incomplete type.
E0521280	[Message]	Conversion from inaccessible base class *xxx* is not allowed.
E0521282	[Message]	String literals with different character kinds cannot be concatenated.
E0521291	[Message]	A non-POD class type cannot be fetched by va_arg.
E0521292	[Message]	The 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal.
E0521295	[Message]	Fixed-point constant is out of range.
E0521303	[Message]	Expression must have integral, enum, or fixed-point type.
E0521304	[Message]	Expression must have integral or fixed-point type.
E0521311	[Message]	Fixed-point types have no classification.
E0521312	[Message]	A template parameter may not have fixed-point type.
E0521313	[Message]	Hexadecimal floating-point constants are not allowed.
E0521315	[Message]	Floating-point value does not fit in required fixed-point type.
E0521317	[Message]	Fixed-point conversion resulted in a change of sign.
E0521318	[Message]	Integer value does not fit in required fixed-point type.
E0521319	[Message]	Fixed-point operation result is out of range.
E0521320	[Message]	Multiple named address spaces.
E0521321	[Message]	Variable with automatic storage duration cannot be stored in a named address space.
E0521322	[Message]	Type cannot be qualified with named address space.
E0521323	[Message]	Function type cannot be qualified with named address space.

E0521324	[Message]	Field type cannot be qualified with named address space.
E0521325	[Message]	Fixed-point value does not fit in required floating-point type.
E0521326	[Message]	Fixed-point value does not fit in required integer type.
E0521327	[Message]	Value does not fit in required fixed-point type.
E0521344	[Message]	A named address space qualifier is not allowed here.
E0521345	[Message]	An empty initializer is invalid for an array with unspecified bound.
E0521348	[Message]	Declaration hides "*symbol*".
E0521349	[Message]	A parameter cannot be allocated in a named address space.
E0521350	[Message]	Invalid suffix on fixed-point or floating-point constant.
E0521351	[Message]	A register variable cannot be allocated in a named address space.
E0521352	[Message]	Expected "SAT" or "DEFAULT".
E0521355	[Message]	A function return type cannot be qualified with a named address space.
E0521365	[Message]	Named-register variables cannot have void type.
E0521372	[Message]	Nonstandard qualified name in global scope declaration.
E0521380	[Message]	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
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.
E0521382	[Message]	Expression must have fixed-point type.
E0521398	[Message]	Invalid member for anonymous member class -- class *xxx* has a disallowed member function.
E0521403	[Message]	A variable-length array is not allowed in a function return type.
E0521404	[Message]	Variable-length array type is not allowed in pointer to member of type "*type*".
E0521405	[Message]	The result of a statement expression cannot have a type involving a variable-length array.
E0521420	[Message]	Some enumerator values cannot be represented by the integral type underlying the enum type.
E0521424	[Message]	Second operand of offsetof must be a field.
E0521425	[Message]	Second operand of offsetof may not be a bit field.
E0521436	[Message]	*xxx* is only allowed in C.
E0521437	[Message]	__ptr32 and __ptr64 must follow a "*".
E0521441	[Message]	Complex integral types are not supported.
E0521442	[Message]	__real and __imag can only be applied to complex values.
E0521445	[Message]	Invalid redefinition of "*symbol*".
E0521534	[Message]	Duplicate function modifier.
E0521535	[Message]	Invalid character for char16_t literal.
E0521536	[Message]	__LPREFIX cannot be applied to char16_t or char32_t literals.

E0521537	[Message]	Unrecognized calling convention *xxx* must be one of:
E0521539	[Message]	Option "--uliterals" can be used only when compiling C.
E0521542	[Message]	Some enumerator constants cannot be represented by "type".
E0521543	[Message]	*xxx* not allowed in current mode.
E0521557	[Message]	Alias creates cycle of aliased entities.
E0521558	[Message]	Subscript must be constant.
E0521574	[Message]	Static assertion failed with *xxx*.
E0521576	[Message]	Field name resolves to more than one offset -- see "*symbol1*" and "*symbol2*".
E0521577	[Message]	*xxx* is not a field name.
E0521578	[Message]	case label value has already appeared in this switch at line *number*.
E0521582	[Message]	The option to list macro definitions may not be specified when compiling more than one translation unit.
E0521583	[Message]	Unexpected parenthesis after declaration of "symbol" (malformed parameter list or invalid initializer?).
E0521584	[Message]	Parentheses around a string initializer are nonstandard.
E0521586	[Message]	A variable declared with an auto type specifier cannot appear in its own initializer.
E0521587	[Message]	Cannot deduce "auto" type.
E0521588	[Message]	Initialization with "{...}" is not allowed for "auto" type.
E0521589	[Message]	auto type cannot appear in top-level array type.
E0521590	[Message]	auto type cannot appear in top-level function type.
E0521593	[Message]	Cannot deduce "auto" type (initializer required).
E0521596	[Message]	Invalid use of a type qualifier.
E0521597	[Message]	A union cannot be abstract or sealed.
E0521598	[Message]	auto is not allowed here.
E0521602	[Message]	struct/union variable "*variable*" with a member of incomplete type cannot be placed into the section.
E0521603	[Message]	Variable of incomplete type "*variable*" cannot be placed into the section.
E0521604	[Message]	Illegal section attribute.
E0521605	[Message]	Illegal #pragma *character string* syntax.
E0521606	[Message]	"*function*" has already been placed into another section.
	[Explanation]	A "#pragma text" has already been specified for function "*function*". It cannot be put into a different section.
E0521608	[Message]	#pragma asm is not allowed outside of function.
E0521609	[Message]	The #pragma endasm for this #pragma asm is missing.
E0521610	[Message]	The #pragma asm for this #pragma endasm is missing.
E0521612	[Message]	Duplicate interrupt hander for "*request*".
E0521613	[Message]	Interrupt request name "*request*" not supported.
E0521614	[Message]	Duplicate #pragma interrupt for this function.

E0521615	[Message]	Duplicate #pragma smart_correct for this function "*function*".
	[Explanation]	A "#pragma smart_correct" has already been specified for function "*function*".
E0521616	[Message]	*Type* "*symbol*" has already been placed into another section (declared as extern).
E0521617	[Message]	*Type* "*symbol*" has already been placed into another section.
E0521618	[Message]	*Type* "*symbol*" has already been declared with #pragma section.
E0521619	[Message]	*Type* "*symbol*" has already been declared without #pragma section.
E0521620	[Message]	"*function*()" argument overflow. use "*minimum value - maximum value*".
E0521621	[Message]	Cannot write I/O register "*register name*".
E0521622	[Message]	Cannot read I/O register "*register name*".
E0521623	[Message]	Cannot use *expanded specification*. Device must be specified.
E0521624	[Message]	Second argument for __set_il()must be string literal.
E0521625	[Message]	Cannot set interrupt level for "*request*".
E0521626	[Message]	*Specification character string* is specified for function "*function name*", previously specified #pragma inline is ignored.
E0521627	[Message]	Function for #pragma smart_correct is same.
E0521628	[Message]	Function for #pragma smart_correct "*function*" is undefined.
E0521629	[Message]	Could not open symbol file "*file name*".
E0521630	[Message]	Could not close symbol file "*file name*".
E0521631	[Message]	Syntax error in symbol file.
E0521632	[Message]	Unrecognized symbol information "*character string*" is ignored.
E0521633	[Message]	Section name is not specified.
E0521634	[Message]	Unrecognized section name "*section*".
E0521635	[Message]	"*variable name*" has already been placed into "*section name*" section in symbol file. The latter is ignored.
E0521636	[Message]	"*variable name*" has already been placed into "*section name*" section in symbol file. #pragma is ignored.
E0521637	[Message]	Illegal binary digit.
E0521638	[Message]	First argument for *special function name*()must be integer constant.
E0521639	[Message]	Function "*function name*" specified as "direct" can not be allocated in text.
E0521640	[Message]	Function allocated in text can not be specified #pragma interrupt with "direct".
E0521641	[Message]	FE level interrupt not supported.
E0521642	[Message]	Cannnot give a name for "*attribute*" section.
E0521643	[Message]	"direct" cannot be specified for plural interrupt.
E0521644	[Message]	Reduced exception handler option of device is available. Address of the handler-maybe overlaps.
E0521645	[Message]	Function "*function name*" has illegal type for interrupt function,must be void(void).
E0521646	[Message]	Cannot use direct with NO_VECT.
E0521647	[Message]	*character string* is not allowed here.

E0521648	[Message]	Cannot call *type* function "*function name*".
E0521649	[Message]	Cannot use *character string1* with *character string2*.
	[Explanation]	The functions of *string1* and *string2* cannot be used at the same time.
	[Message]	[CC-RX] White space is required between the macro name *xxx* and its replacement text.
	[Action by User]	[CC-RX] Insert white space between the macro name and its replacement text.
E0521650	[Message]	*type* "*symbol name*" has already been declared with other #pragma pic/nopic.
	[Explanation]	There is a "#pragma pin/nopic" specification in conflict with *type* "*symbol name*".
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.
E0523018	[Message]	a member qualified with near or far is declared
	[Explanation]	__near or __far cannot be specified for a member when defining a structure or union.
E0523042	[Message]	Using "*function item*" function at influence the code generation of "SuperH" compiler
	[Explanation]	The use of "*function item*" may affect compatibility with the SuperH compiler. Confirm details of differences from the specification.
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.
E0523059	[Message]	Cannot change text section
	[Explanation]	The #pragma section syntax is incorrect.
E0523061	[Message]	Argument is incompatible with formal parameter of intrinsic function.
E0523062	[Message]	Return value type does not match the intrinsic function type.
E0523065	[Message]	[CC-RX] "*character string*" has no effect in this version
	[Message]	[CC-RH] Cannot assign address constant to initializer for bitfield
	[Action by User]	[CC-RH] Do not write an address constant as the initial value of the bit field.
E0523066	[Message]	The combination of the option and section specification is inaccurate

E0523067	[Message]	Type nest is too deep
	[Explanation]	Nesting of the declarator is too deep.
	[Action by User]	Do not write nesting that exceeds the limit of the processing system.
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
E0523074	[Message]	"*function name*" cannot be used with #pragma rtos_interrupt
	[Explanation]	#pragma rtos_interrupt cannot be specified for "*function name*".
E0523075	[Message]	Combination of address and near/far attribute is incorrect
	[Explanation]	The address specified by #pragma address is a location conflicting with the attributes of __near and __far which are specified as variables.
E0532002	[Message]	Exception *exception* has occurred at compile time.
E0541004	[Message]	Addition/subtraction of __sectop/__secend and a constant are not allowed.
E0541240	[Message]	Illegal naming of section "*section name*".
E0541854	[Message]	Illegal address was specified with #pragma address.
	[Explanation]	The same address is specified for different variables.
E0544003	[Message]	The size of "*section name*" section exceeds the limit.
E0544240	[Message]	Illegal naming of section "*section name*".
	[Explanation]	There is an error in section naming. The same section name is specified for different use of the section.
E0544854	[Message]	Illegal address was specified with #pragma address.
	[Explanation]	"#pragma address" specification satisfies either of the following conditions. (1) The same address was specified for different variables. (2) Overlapping address ranges were specified for different variables.
E0550200	[Message]	Illegal alignment value.
	[Action by User]	Check the alignment condition specification.
E0550201	[Message]	Illegal character.
	[Action by User]	Check the character.
E0550202	[Message]	Illegal expression.
	[Action by User]	Check the expression.
E0550203	[Message]	Illegal expression (*string*).
	[Action by User]	Check the expression element.
E0550204	[Message]	Illegal expression (-label).
	[Action by User]	Check the expression.
E0550205	[Message]	Illegal expression (-label - label).
	[Action by User]	Check the expression.

E0550206	[Message]	Illegal expression (label + label).
	[Action by User]	Check the expression.
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.
E0550210	[Message]	Illegal expression (not + nor -).
	[Action by User]	Check the expression.
E0550211	[Message]	Floating exception(*function*).
	[Action by User]	Check the results of the operation.
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.
E0550215	[Message]	Illegal operand (access width mismatch).
	[Action by User]	Check the internal peripheral I/O register.
E0550216	[Message]	Illegal operand (cannot read I/O register which does not have read access).
	[Action by User]	Check the internal peripheral I/O register.
E0550217	[Message]	Illegal operand (cannot use bit I/O register).
	[Action by User]	Check the internal peripheral I/O register.
E0550218	[Message]	Illegal operand (cannot write I/O register which does not have write access).
	[Action by User]	Check the internal peripheral I/O register.
E0550219	[Message]	Illegal operand (inconsistent bit position).
	[Action by User]	Check the bit location.
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.
E0550222	[Message]	Illegal operand (label not allowed).
	[Action by User]	Check the operand.
E0550223	[Message]	Illegal operand (label not allowed for setf/shl...).
	[Action by User]	Check the operand.

E0550224	[Message]	Illegal operand (label reference for jmp must be *string*).
	[Action by User]	Check the operand.
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.
E0550227	[Message]	Illegal operand (must be immediate, label or symbol for operator).
	[Action by User]	Check the operand.
E0550228	[Message]	Illegal operand (must be register).
	[Action by User]	Check the operand.
E0550229	[Message]	Illegal operand (needs base resister).
	[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.
E0550233	[Message]	Illegal operand (local symbol 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 *device-name* mode).
	[Action by User]	Check the operand.
E0550240	[Message]	Illegal operand (cannot use r0 as destination in *device-name* mode).
	[Action by User]	Check the operand.
E0550241	[Message]	Illegal operand (too many registers).
	[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.
E0550251	[Message]	Illegal ID value.
	[Explanation]	Illegal ID value. Specify an integer value.
	[Action by User]	Check the ID value.
E0550252	[Message]	ID already defined as symbol "*identifier*".
	[Action by User]	Check the ID value.
E0550253	[Message]	Symbol "*identifier*" already defined as another ID.
	[Action by User]	Check the ID value.
E0550254	[Message]	Cannot reference $EXT_FUNC symbol "*identifier*".
	[Action by User]	Check the code.
E0550255	[Message]	Cannot access for I/O register bit number *"I/O register*".
	[Action by User]	Check the bit number.
E0550258	[Message]	Cannot access I/O register("*I/O register*").
	[Action by User]	Check the operand.
E0550259	[Message]	Cannot use r1 as destination in mul/mulu.
	[Action by User]	Check the destination register.
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]	[RH850] Illegal condition code. 0xd cannot be specified for the condition code of an adf.sbf instruction [V850E2].
	[Action by User]	Check the condition code.
E0550262	[Message]	Illegal operand (cannot use r0 as destination in *device-name* mode).
	[Action by User]	Check the operand.
E0550263	[Message]	Illegal operand (cannot use $label,%label in common area).
	[Action by User]	Check the operand.
E0550264	[Message]	Illegal section kind (common area).
	[Action by User]	Check the section kind.
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.
E0550266	[Message]	Illegal vector register number (vr16-vr31).
	[Action by User]	You can only specify one of vr0 to vr15 as the vector register. Check the operand.
E0550267	[Message]	Illegal operand (displacement must be multiple of 4).
	[Action by User]	Check the displacement.
E0550268	[Message]	Illegal vector register number (vr0-vr23).
	[Explanation]	Only vr24-vr31 are specifiable as vector registers.
	[Action by User]	Check the operand.
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.
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.
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.
E0550618	[Message]	Symbol is required for the "*character string*" option.
E0550620	[Message]	Specify a value (*value1-value2*) for the "*character string*" option.
E0550621	[Message]	Multiple source files are not allowed when both the "-o" option and the "*character string*" option are specified.
E0550622	[Message]	The argument for the "*character string*" optionmust be an object file.
E0550624	[Message]	The "-Xcommon" option must be specified.
E0550625	[Message]	Cannot find device file.
E0550626	[Message]	Device file "*file-name*" read error.
E0550629	[Message]	Command file "*file-name*" is readmore than once.
E0550630	[Message]	Command file "*file-name*" cannot be read.
E0550631	[Message]	Syntax error in command file "*file-name*".
E0550632	[Message]	Failed to create temporary folder.
E0550634	[Message]	Input file "*file-name*" is not found.
E0550635	[Message]	"*path-name*" specified as an input file is a folder.
E0550636	[Message]	Failed to delete a temporary file "*file-name*".
E0550642	[Message]	Multiple source files are not allowed when the "*character string*" option is specified.
E0550645	[Message]	"*character string2*" specified in the "*character string1*" option is not available.
E0550648	[Message]	"*file-name*" is specified as an output file for the different options.
	[Action by User]	Specify another filename.
E0550650	[Message]	The "*character string1*" option and the "*character string2*" option are inconsistent.
E0550652	[Message]	The "*character string1*" option needs the "*character string2*" option.
E0550654	[Message]	Component file "*file-name*" for the CC-RH is not found. Reinstall the CC-RH.
E0550661	[Message]	Failed to delete a file "*file-name*".
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.
E0551201	[Message]	Expected a ","
	[Action by User]	Check whether a comma is missing.
E0551202	[Message]	Illegal register.
	[Explanation]	There is a register that cannot be specified as an operand.
	[Action by User]	Check which registers can be specified as operands.

E0551203	[Message]	Relocatable symbol is not allowed.
	[Explanation]	There is a relocatable symbol at a location not allowed.
	[Action by User]	Check the description format of the respective location.
E0551204	[Message]	Illegal operands.
	[Explanation]	An illegal operand is specified.
	[Action by User]	Check the formats that can be specified as operands.
E0551205	[Message]	Illegal string.
	[Explanation]	There is an error in the string.
	[Action by User]	Check if there are errors in the string.
E0551206	[Message]	"$" is not allowed.
	[Explanation]	There is "$" where it is not allowed.
	[Action by User]	Check that there is no "$" where it is not allowed.
E0551207	[Message]	"*string*" is not allowed.
	[Action by User]	Check the description format of the respective location.
E0551208	[Message]	Illegal operation ("op").
	[Explanation]	There is an error in the description of "op" operation.
	[Action by User]	Check the description of "op" operation.
E0551209	[Message]	Illegal 1st operand in bit position specifier.
	[Action by User]	Check the description of the 1st operand of the bit position specifier.
E0551210	[Message]	Byte separation operator for bit reference is not allowed.
	[Action by User]	Apply the separation operator to the 1st operand for bit reference.
E0551211	[Message]	Only bit symbols are allowed.
	[Action by User]	Check that there are only bit symbols.
E0551212	[Message]	Illegal bit position specifier.
	[Action by User]	Check the description of the bit position specifier.
E0551213	[Message]	Operand or right parenthesis is missing.
	[Explanation]	Either a right parenthesis is missing or there is no expression to be targeted by the operator.
	[Action by User]	Check that there is a right parenthesis to match each left parenthesis or there is an expression to be targeted by the operator.
E0551214	[Message]	Illegal operation ("op").
	[Action by User]	Check the format of the op operator.
E0551215	[Message]	Illegal label reference.
	[Action by User]	Check the description of the label.
E0551216	[Message]	Illegal expression (label - label).
	[Explanation]	There is an expression of the (label - label) format where it is not allowed.
	[Action by User]	Check the expression.

E0551217	[Message]	Illegal expression (label + label).
	[Explanation]	An expression of the (label + label) format is not allowed.
	[Action by User]	Check the expression.
E0551218	[Message]	Illegal expression (-label).
	[Explanation]	An expression of the (-label) format is not allowed.
	[Action by User]	Check the expression.
E0551219	[Message]	Illegal label reference.
	[Explanation]	Operation or reference of a label is invalid.
	[Action by User]	Check the Operation or reference of a label.
E0551220	[Message]	Undefined symbol is not allowed.
	[Explanation]	There is an undefined symbol where it is not allowed.
	[Action by User]	Check the symbol definition.
E0551221	[Message]	Section name is not allowed.
	[Explanation]	There is a section name where it is not allowed.
	[Action by User]	Check which section names are allowed.
E0551222	[Message]	Illegal character.
	[Explanation]	Failed to read characters.
	[Action by User]	Check the code.
E0551223	[Message]	Closing single quotation mark is missing.
	[Explanation]	A single quotation (') is not closed.
	[Action by User]	Check the single quotation (') is not closed.
E0551224	[Message]	Illegal string.
	[Explanation]	Failed to read strings.
	[Action by User]	Check the code.
E0551225	[Message]	Closing double quotation mark is missing.
	[Explanation]	A double quotation (") is not closed.
	[Action by User]	Check if the double quotation (") is closed.
E0551226	[Message]	Illegal string in expression.
	[Explanation]	There is a string in the middle of an expression.
E0551227	[Message]	'?' is not allowed.
	[Explanation]	'?' is not handled as an alphanumeric character. It cannot be used in a symbol name.
E0551228	[Message]	Numeric description does not match -base_number option.
E0551229	[Message]	Invalid binary number.
	[Action by User]	Check if the binary notation is correct.
E0551230	[Message]	Invalid octal number.
	[Action by User]	Check if the octal notation is correct.

E0551231	[Message]	Invalid decimal number.
	[Action by User]	Check if the decimal notation is correct.
E0551232	[Message]	Invalid hexadecimal number.
	[Action by User]	Check if the hexadecimal notation is correct.
E0551233	[Message]	Too many operands.
	[Action by User]	Specify operands for the correct number.
E0551234	[Message]	Closing bracket is missing.
	[Explanation]	There is no right bracket.
E0551235	[Message]	Nesting of this oprator is not allowed.
E0551236	[Message]	Illegal tilde operation.
	[Explanation]	There is an error in the description of the tilde.
E0551301	[Message]	Bit number should.be in the range 0-7.
E0551302	[Message]	Specified address is out of saddr area.
	[Explanation]	The value specified for the operand is outside of the saddr area.
E0551303	[Message]	Specified address is out of SFR area.
	[Explanation]	The value specified for the operand is outside of the sfr area.
E0551304	[Message]	Specified address is out of callt table area.
	[Explanation]	The value specified for the operand is outside of the callt table area.
E0551305	[Message]	Specified value is out of 8-bit integer.
	[Explanation]	The value specified for the operand exceeds the 8-bit width.
E0551306	[Message]	Specified value is out of 16-bit integer.
	[Explanation]	The value specified for the operand exceeds the 16-bit width.
E0551307	[Message]	Specified value is out of 20-bit integer.
	[Explanation]	The value specified for the operand exceeds the 20-bit width.
E0551308	[Message]	Specified value is out of 32-bit integer.
	[Explanation]	The value specified for the operand exceeds the 32-bit width.
E0551309	[Message]	Odd number is now allowed.
E0551310	[Message]	Only "1" is allowed.
E0551311	[Message]	Specified value is out of range 0-7.
E0551312	[Message]	Specified value is out of range 1-15.
E0551313	[Message]	"*reg*"is not allowed.
	[Explanation]	Register *reg* is not allowed here.
	[Action by User]	Check which operands are allowed.
E0551314	[Message]	Only HL register is allowed.
E0551315	[Message]	Only ES register is allowed.

E0551316	[Message]	Illegal SFR or control register.
	[Explanation]	There is an illegal SFR or a control register.
E0551317	[Message]	Forward reference of SFR is not allowed.
E0551401	[Message]	Illegal operand "*operand*".
	[Explanation]	There is an error in the description of the operand.
	[Action by User]	Check the description of the operand.
E0551402	[Message]	Illegal instruction.
	[Explanation]	The instruction type is illegal.
E0551403	[Message]	Illegal operand of .DB8 directive.
	[Explanation]	A separation operator cannot be set for the operand of the .DB8 pseudo instruction.
	[Action by User]	Check if the .DB8 operand is described correctly.
E0551404	[Message]	Illegal address description of .VECTOR directive.
	[Action by User]	Check if the address of the .VECTOR pseudo instruction is specified correctly.
E0551405	[Message]	Illegal $MIRROR declaration.
	[Explanation]	$MIRROR cannot be specified for the symbol.
	[Action by User]	Check whether the $MIRROR pseudo instruction specifies an external reference name.
E0551501	[Message]	Multiple source files are not allowed when the "-output" option is specified.
E0552000	[Message]	No space after mnemonic or directive.
	[Explanation]	The mnemonic or assemble directive is not followed by a space character.
	[Action by User]	Enter a space character between the instruction and operand.
E0552001	[Message]	',' is missing.
	[Explanation]	',' is not entered.
	[Action by User]	Insert a comma to separate between operands.
E0552002	[Message]	Characters exist in expression.
	[Explanation]	Extra characters are written in an instruction or expression.
	[Action by User]	Check the rules to be followed when writing an expression.
E0552003	[Message]	Size specifier is missing.
	[Explanation]	No size specifier is entered.
	[Action by User]	Write a size specifier.
E0552004	[Message]	Invalid operand(s) exist in instruction.
	[Explanation]	The instruction contains an invalid operand.
	[Action by User]	Check the syntax for this instruction and rewrite it correctly.
E0552005	[Message]	Operand type is not appropriate.
	[Explanation]	The operand type is incorrect.
	[Action by User]	Check the syntax for this operand and rewrite it correctly.

E0552006	[Message]	Size specifier is not appropriate.
	[Explanation]	The size specifier is written incorrectly.
	[Action by User]	Rewrite the size specifier correctly.
E0552007	[Message]	Operand label is not in the same section.
	[Explanation]	The branch destination is not in the same section.
	[Action by User]	Execution can branch only to a destination within the same section. Correct the mnemonic.
E0552008	[Message]	Illegal displacement value.
	[Explanation]	An illegal displacement value is specified.
	[Action by User]	Specify a multiple of 2 when the size specifier is W. Specify a multiple of 4 when the size specifier is L.
E0552009	[Message]	FPU instruction or FPSW is used.
	[Explanation]	A floating-point operation (FPU) instruction or FPSW is used.
	[Action by User]	Check the CPU type.
E0552010	[Message]	ISAV2 instruction or EXTB is used
E0552022	[Message]	Symbol name is missing.
	[Explanation]	Symbol is not entered.
	[Action by User]	Write a symbol name.
E0552023	[Message]	Illegal directive command is used.
	[Explanation]	An illegal instruction is entered.
	[Action by User]	Rewrite the instruction correctly.
E0552024	[Message]	No ';' at the top of comment.
	[Explanation]	';' is not entered at the beginning of a comment.
	[Action by User]	Enter a semicolon at the beginning of each comment. Check whether the mnemonic or operand is written correctly.
E0552026	[Message]	'CODE' section in big endian is not appropriate.
	[Explanation]	The value specified for the start address of the absolute-addressing CODE section is not a multiple of 4 while endian=big is specified.
	[Action by User]	Specify a multiple of 4 for the start address.
E0552027	[Message]	Illegal character code.
	[Explanation]	An illegal character code is specified.
E0552028	[Message]	Unrecognized character escape sequence.
	[Explanation]	An unrecognizable escape sequence is specified.
E0552029	[Message]	Invalid description in #pragma inline_asm function.
	[Explanation]	Invalid assembly-language code was usedin an assembly-language function. Go through the C-language source file and check the code corresponding to functions for which #pragma_inline_asm was specified.

E0552040	[Message]	Include nesting over.
	[Explanation]	Include is nested too many levels.
	[Action by User]	Rewrite include so that it is nested within 30 levels.
E0552041	[Message]	Can't open include file '*XXXX*'.
	[Explanation]	The include file cannot be opened.
	[Action by User]	Check the include file name. Check the directory where the include file is stored.
E0552042	[Message]	Including the include file in itself.
	[Explanation]	An attempt is made to include the include file in itself.
	[Action by User]	Check the include file name and rewrite correctly.
E0552049	[Message]	Invalid reserved word exist in operand.
	[Explanation]	The operand contains a reserved word.
	[Action by User]	Reserved words cannot be written in an operand. Rewrite the operand correctly.
E0552050	[Message]	Operand value is not defined.
	[Explanation]	An undefined operand value is entered.
	[Action by User]	Write a valid value for operands.
E0552051	[Message]	'{' is missing.
	[Explanation]	'{' is not specified.
E0552052	[Message]	Addressing mode specifier is not appropriate.
	[Explanation]	The addressing mode specifier is written incorrectly.
	[Action by User]	Make sure that the addressing mode is written correctly.
E0552053	[Message]	Reserved word is missing.
	[Explanation]	No reserved word is entered.
E0552054	[Message]	']' is missing.
	[Explanation]	']' is not entered.
	[Action by User]	Write the right bracket ']' corresponding to the '['.
E0552055	[Message]	Right quote is missing.
	[Explanation]	A right quote is not entered.
	[Action by User]	Enter the right quote.
E0552056	[Message]	The value is not constant.
	[Explanation]	The value is indeterminate when assembled.
	[Action by User]	Write an expression, symbol name, or label name that will have a determinate value when assembled.
E0552057	[Message]	Quote is missing.
	[Explanation]	Quotes for a character string are not entered.
	[Action by User]	Enclose a character string with quotes as you write it.

E0552058	[Message]	Illegal operand is used.
	[Explanation]	The operand is incorrect.
	[Action by User]	Check the syntax for this operand and rewrite it correctly.
E0552059	[Message]	Operand number is not enough.
	[Explanation]	The number of operands is insufficient.
	[Action by User]	Check the syntax for these operands and rewrite them correctly.
E0552060	[Message]	Too many macro nesting.
	[Explanation]	The macro is nested too many levels.
	[Action by User]	Make sure that the macro is nested no more than 65,535 levels. Check the syntax for this source statement and rewrite it correctly.
E0552061	[Message]	Too many macro local label definition.
	[Explanation]	Too many macro local labels are defined.
	[Action by User]	Make sure that the number of macro local labels defined in one file are 65,535 or less.
E0552062	[Message]	'.MACRO' is missing for '.ENDM'.
	[Explanation]	.MACRO for .ENDM is not found.
	[Action by User]	Check the position where .ENDM is written.
E0552063	[Message]	'.MREPEAT' is missing for '.ENDR'.
	[Explanation]	.MREPEAT for .ENDR is not found.
	[Action by User]	Check the position where .ENDR is written.
E0552064	[Message]	'.MACRO' or '.MREPEAT' is missing for '.EXITM'.
	[Explanation]	.MACRO or .MREPEAT for .EXITM is not found.
	[Action by User]	Check the position where .EXITM is written.
E0552065	[Message]	No macro name.
	[Explanation]	No macro name is entered.
	[Action by User]	Write a macro name for each macro definition.
E0552066	[Message]	Too many formal parameter.
	[Explanation]	There are too many formal parameters defined for the macro.
	[Action by User]	Make sure that the number of formal parameters defined for the macro is 80 or less.
E0552067	[Message]	Illegal macro parameter.
	[Explanation]	The macro parameter contains some incorrect description.
	[Action by User]	Check the written contents of the macro parameter.
E0552068	[Message]	Source line is too long.
	[Explanation]	The source line is excessively long.
	[Action by User]	Check the contents written in the source line and correct it as necessary.

E0552069	[Message]	'.MACRO' is missing for '.LOCAL'.
	[Explanation]	.MACRO for .LOCAL is not found.
	[Action by User]	Check the position where .LOCAL is written. .LOCAL can only be written in a macro block.
E0552070	[Message]	No '.ENDM' statement.
	[Explanation]	.ENDM is not entered.
	[Action by User]	Check the position where .ENDM is written. Write .ENDM as necessary.
E0552071	[Message]	No '.ENDR' statement.
	[Explanation]	.ENDR is not entered.
	[Action by User]	Check the position where .ENDR is written. Write .ENDR as necessary.
E0552072	[Message]	')' is missing.
	[Explanation]	')' is not entered.
	[Action by User]	Write the right parenthesis ')' corresponding to the '('.
E0552073	[Message]	Operand expression is not completed.
	[Explanation]	The operand description is not complete.
	[Action by User]	Check the syntax for this operand and rewrite it correctly.
E0552074	[Message]	Syntax error in expression.
	[Explanation]	The expression is written incorrectly.
	[Action by User]	Check the syntax for this expression and rewrite it correctly.
E0552075	[Message]	String value exist in expression.
	[Explanation]	A character string is entered in the expression.
	[Action by User]	Rewrite the expression correctly.
E0552076	[Message]	Division by zero.
	[Explanation]	A divide by 0 operation is attempted.
	[Action by User]	Rewrite the expression correctly.
E0552077	[Message]	No '.END' statement.
	[Explanation]	.END is not entered.
	[Action by User]	Be sure to enter .END in the last line of the source program.
E0552078	[Message]	The specified address overlaps at '*address*'.
	[Explanation]	Something has already beenallocated to '*address*'.
	[Action by User]	Check the specifications for .ORG and .OFFSET.
E0552080	[Message]	'.IF' is missing for '.ELSE'.
	[Explanation]	.IF for .ELSE is not found.
	[Action by User]	Check the position where .ELSE is written.
E0552081	[Message]	'.IF' is missing for '.ELIF'.
	[Explanation]	.IF for .ELIF is not found.
	[Action by User]	Check the position where .ELIF is written.

E0552082	[Message]	'.IF' is missing for '.ENDIF'.
	[Explanation]	.IF for .ENDIF is not found.
	[Action by User]	Check the position where .ENDIF is written.
E0552083	[Message]	Too many nesting level of condition assemble.
	[Explanation]	Condition assembling is nested too many levels.
	[Action by User]	Check the syntax for this condition assemble statement and rewrite it correctly.
E0552084	[Message]	No '.ENDIF' statement.
	[Explanation]	No corresponding .ENDIF is found for the .IF statement in the source file.
	[Action by User]	Check the source description.
E0552088	[Message]	Can't open '.ASSERT' message file '*XXXX*'.
	[Explanation]	.The .ASSERT output file cannot be opened.
	[Action by User]	Check the file name.
E0552089	[Message]	Can't write '.ASSERT' message file '*XXXX*'.
	[Explanation]	Data cannot be written to the .ASSERT output file.
	[Action by User]	Check the permission of the file.
E0552090	[Message]	Too many temporary label.
	[Explanation]	There are too many temporary labels.
	[Action by User]	Replace the temporary labels with label names.
E0552091	[Message]	Temporary label is undefined.
	[Explanation]	The temporary label is not defined yet.
	[Action by User]	Define the temporary label.
E0552100	[Message]	Value is out of range.
	[Explanation]	The value is out of range.
	[Action by User]	Write a value that matches the register bit length.
E0552112	[Message]	Symbol is missing.
	[Explanation]	Symbol is not entered.
	[Action by User]	Write a symbol name.
E0552113	[Message]	Symbol definition is not appropriate.
	[Explanation]	The symbol is defined incorrectly.
	[Action by User]	Check the method for defining this symbol and rewrite it correctly.
E0552114	[Message]	Symbol has already defined as another type.
	[Explanation]	The symbol has already been defined in a different directive with the same name.
	[Action by User]	Change the symbol name.
E0552115	[Message]	Symbol has already defined as the same type.
	[Explanation]	The symbol has already been defined.
	[Action by User]	Change the symbol name.

E0552116	[Message]	Symbol is multiple defined.
	[Explanation]	The symbol is defined twice or more. The macro name and some other name are duplicates.
	[Action by User]	Change the symbol name.
E0552117	[Message]	Invalid label definition.
	[Explanation]	An invalid label is entered.
	[Action by User]	Rewrite the label definition.
E0552118	[Message]	Invalid symbol definition.
	[Explanation]	An invalid symbol is entered.
	[Action by User]	Rewrite the symbol definition.
E0552119	[Message]	Reserved word is used as label or symbol.
	[Explanation]	Reserved word is used as a label or symbol.
	[Action by User]	Rewrite the label or symbol name correctly.
E0552130	[Message]	No '.SECTION' statement.
	[Explanation]	.SECTION is not entered.
	[Action by User]	Always make sure that the source program contains at least one .SECTION.
E0552131	[Message]	Section type is not appropriate.
	[Action by User]	An instruction or a directive used in a section does not match the section type.
E0552132	[Message]	Section has already determined as attribute.
	[Explanation]	The attribute of this section has already been defined as relative. Directive command .ORG cannot be written here.
	[Action by User]	Check the attribute of the section.
E0552133	[Message]	Section attribute is not defined.
	[Explanation]	Section attribute is not defined. Directive command .ALIGN cannot be written in this section.
	[Action by User]	Make sure that directive .ALIGN is written in an absolute attribute section or a relative attribute section where ALIGN is specified.
E0552134	[Message]	Section name is missing.
	[Explanation]	No section name is entered.
	[Action by User]	Write a section name in the operand.
E0552135	[Message]	'ALIGN' is multiple specified in '.SECTION'.
	[Explanation]	Two or more ALIGN's are specified in the .SECTION definition line.
	[Action by User]	Delete extra ALIGN specifications.
E0552136	[Message]	Section type is multiple specified.
	[Explanation]	Section type is specified two or more times in the section definition line.
	[Action by User]	Only one section type CODE, DATA, or ROMDATA can be specified in a section definition line.

E0552137	[Message]	Too many operand.
	[Explanation]	There are extra operands.
	[Action by User]	Check the syntax for these operands and rewrite them correctly.
E0560601	[Message]	';' is expected at the end of directive.
E0560602	[Message]	'}' is expectedis expected at end of region.
E0560603	[Message]	Name is expected at the beginning of directive.
E0560604	[Message]	Section name is expected at the beginning of section directive.
E0560605	[Message]	':' , '=' or '@' is expected to follow name.
E0560606	[Message]	'=' is expected to follow section name.
E0560607	[Message]	Too many '}'.
E0560608	[Message]	Illegal character (*number*).
E0560609	[Message]	"*stirng*" needs effective parameter.
E0560610	[Message]	Illegal string in segment directive.
E0560611	[Message]	Illegal string in section directive.
E0560612	[Message]	Illegal string in symbol directive.
E0560613	[Message]	"*stirng*" is illegal in file specification field
E0560614	[Message]	"*stirng*" is illegal in file segment name field
E0560615	[Message]	"*stirng*" specified to segment "*segment*" more than once in same or other directive.
E0560616	[Message]	"*stirng*" specified to section "*section*" more than once in same or other directive.
E0560617	[Message]	"*stirng*" specified to symbol "*symbol*" more than once in same or other directive.
E0560618	[Message]	Segment "*segment*" already defined.
E0560619	[Message]	Section "*section*" already defined at line(*number*).
E0560620	[Message]	Symbol "*symbol*" already defined at line(*number*).
E0560621	[Message]	Illegal segment type "*stirng*".
E0560622	[Message]	Illegal section type "*stirng*".
E0560623	[Message]	Illegal section attribute '*character*'.
E0560624	[Message]	"*stirng*" in segment directive of non LOAD segment is illegal.
E0560637	[Message]	Unknown symbol kind "*stirng*".
E0560638	[Message]	Symbol kind "*stirng*" specified more than once in same or other directive.
E0560641	[Message]	Section attribute "*attribute*" of section '*section*' and segment attribute '*attribute*' of segment "*segment*" do not match.
E0560660	[Message]	S-JIS code (*number1*, *number2*) is broken.
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.
E0562017	[Message]	Cannot output "*section*" specified in option "*option*"
	[Explanation]	Could not output a portion of the code in "section" specified by "option." Part of the instruction code in "section" has been swapped with instruction code in another section due to endian conversion.
	[Action by User]	Check the section address range with respect to 4-byte boundaries in the linkage list and find which section code is swapped with the target section code.
E0562020	[Message]	Duplicate file specified in option "*option*" : "*file*"
	[Explanation]	The same file was specified twice in *option*.
E0562021	[Message]	Duplicate symbol/section specified in option "*option*" : "*name*"
	[Explanation]	The same symbol name or section name 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*.
E0562112	[Message]	"*section1*" and "*section2*" cannot mapped as ROM/RAM in "*file*"
	[Explanation]	*section1* and *section2* specified in *file* are not ROM/RAM-linked.
E0562113	[Message]	Option "rom" and internal information in the file are conflicted
	[Explanation]	Specification of the -rom option conflicts with the internal information.
E0562120	[Message]	Library "*file*" without module name specified as input file
	[Explanation]	A library file without a module name was specified as the input file.
E0562121	[Message]	Input file is not library file : "*file*(*module*)"
	[Explanation]	The file specified by *file* (*module*) as the input file is not a library file.
E0562130	[Message]	Cannot find file specified in option "*option*" : "*file*"
	[Explanation]	The file specified in *option* could not be found.
E0562131	[Message]	Cannot find module specified in option "*option*" : "*module*"
	[Explanation]	The module specified in *option* could not be found.
E0562132	[Message]	Cannot find "*name*" specified in option "*option*"
	[Explanation]	The symbol or section specified in *option* does not exist.
E0562133	[Message]	Cannot find defined symbol "*name*" in option "*option*"
	[Explanation]	The externally defined symbol specified in *option* does not exist.
E0562134	[Message]	Reserved section name "*section*"
	[Explanation]	" *section*" is the reservation name used by a linker.
	[Action by User]	Check if the section name is correct.
E0562135	[Message]	Interrupt number "*vector-number*" has invalid interrupt jump address : "*symbol*"
	[Explanation]	"*symbol*" cannot be specified as an interrupt function.
	[Action by User]	Check the description of the option and source file.
E0562140	[Message]	Symbol/section "*name*" redefined in option "*option*"
	[Explanation]	The symbol or section specified in *option* has already been defined.
E0562141	[Message]	Module "*module*" redefined in option "*option*"
	[Explanation]	The module specified in *option* has already been defined.

E0562142	[Message]	Interrupt number "*vector number*" of "*section*" has multiple definition
	[Explanation]	Vector number definition was made multiple times in vector table *section*. Only one address can be specified for a vector number.
	[Action by User]	Check and correct the code in the source file.
E0562200	[Message]	Illegal object file : "*file*"
	[Explanation]	A format other than ELF format was input.
E0562201	[Message]	Illegal library file : "*file*"
	[Explanation]	*file* is not a library file.
E0562203	[Message]	Illegal profile information file : "*file*"
	[Explanation]	*file* is not a profile information file.
E0562204	[Message]	Unsupported device file : "*file*"
	[Explanation]	"*file*" cannot be read as the device file.
	[Action by User]	Check the device file.
E0562210	[Message]	Invalid input file type specified for option "*option*" : "*file*(*type*)"
	[Explanation]	When specifying *option*, a *file* (*type*) that cannot be processed was input.
E0562211	[Message]	Invalid input file type specified while processing "*process*" : "*file*(*type*)"
	[Explanation]	A *file* (*type*) that cannot be processed was input during processing *process*.
E0562212	[Message]	"*option*" cannot be specified for inter-module optimization information in "*file*"
	[Explanation]	The option *option* cannot be used because *file* includes inter-module optimization information.
	[Action by User]	Do not specify the goptimize option at compilation or assembly.
E0562220	[Message]	Illegal mode type "*mode type*" in "*file*"
	[Explanation]	A file with a different *mode type* was input.
E0562221	[Message]	Section type mismatch : "*section*"
	[Explanation]	Sections with the same name but different attributes (whether initial values present or not) were input.
E0562223	[Message]	Cpu type "CPU type1" in "file" is incompatible with "CPU type2"
	[Explanation]	A different CPU type is input. Since these types are incompatible in part of specifications, even if the file is linked, behavior cannot be guaranteed.
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.
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.
E0562323	[Message]	Section "*section1*(*address range*)" overlaps with section "*section2*(*address range*)" in physical space
	[Explanation]	*section1* overlaps with *section2* in the physical memory.
	[Action by User]	Check the addresses of the sections. <*address range*>: <*section start address*> - <*section end address*>
E0562324	[Message]	Section "*section*" in "*file*" conflicts
	[Explanation]	More than one object file containing "*section*" was input.
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.
E0562331	[Message]	Division by zero in relocation value calculation : "*file*"-"*section*"-"*offset*"
	[Explanation]	Division by zero occurred during a relocation operation.
	[Action by User]	Check for problems in calculation of the position at *offset* in *section* in the source list.
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.
E0562401	[Message]	near8,near16 symbol "*symbol*" is outside near memory area
	[Explanation]	*symbol* is not allocated in the near8 or near16 range.
	[Action by User]	Either change the start specification, or remove the near specifier at compilation, so that correct address calculations can be made.
E0562402	[Message]	Number of register parameter conflicts with that in another file : "*function*"
	[Explanation]	Different numbers of register parameters are specified for *function* in multiple files.

E0562403	[Message]	Fast interrupt register in "*file*" conflicts with that in another file
	[Explanation]	The register number specified for the fast interrupt general register in *file* does not match the settings in other files.
	[Action by User]	Correct the register number to match the other settings and recompile the code.
E0562404	[Message]	Base register "*base register type*" in "*file*" conflicts with that in another file
	[Explanation]	The register number specified for *base register type* in *file* does not match the settings in other files.
	[Action by User]	Correct the register number to match the other settings and recompile the code.
E0562405	[Message]	Option "*compile option*" conflicts with that in other files
	[Explanation]	Specification of *compile option* is inconsistent between the input files.
	[Action by User]	Review the compile option.
E0562406	[Message]	General-purpose register "*register*" in "*file*" conflicts with that in another file("*mode*")
	[Explanation]	The usage of general-purpose register "*register*" specified in "*file*" is not consistent with the usage in other files.
	[Action by User]	Check the options used on compiling.
E0562407	[Message]	Handler stack pointer "*register*" in "*file*" conflicts with that in another file("*mode*")
	[Explanation]	The usage of handler stack pointer "*register*" specified in "*file*" is not consistent with the usage in other files.
	[Action by User]	Check the options used on compiling.
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. (3) When the tbr option or #pragma tbr is used, optimization by the compiler may delete symbols after the map option specification at compilation. Disable the map option at compilation or disable the tbr option or #pragma tbr.
E0562411	[Message]	Map file in "*file*" conflicts with that in another file
	[Explanation]	Different external symbol allocation information files were used by the input files at compilation.
E0562412	[Message]	Cannot open file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be opened.
	[Action by User]	Check whether the file name and access rights are correct.

E0562413	[Message]	Cannot close file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be closed. There may be insufficient disk space.
E0562414	[Message]	Cannot read file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) cannot be read. There may be insufficient disk space.
E0562415	[Message]	Illegal map file : "*file*"
	[Explanation]	*file* (external symbol allocation information file) has an illegal format.
	[Action by User]	Check whether the file name is correct.
E0562416	[Message]	Order of functions specified by map file differs from one after linkage as to "*function name*"
	[Explanation]	The sequences of a function *function name* and those of other functions are different between the information within the external symbol allocation information file used at compilation and the location after linkage. The address of static within the function may be different between the external symbol allocation information file and the result after linkage.
E0562417	[Message]	Map file is not the newest version: "*file name*"
	[Explanation]	The external symbol allocation information file is not the latest version.
E0562420	[Message]	"*file1*" overlap address "*file2*" : "*address*"
	[Explanation]	The address specified for *file1* is the same as that specified for *file2*.
E0562500	[Message]	Cannot find library file : "*file*"
	[Explanation]	*file* specified as a library file cannot be found.
E0572000	[Message]	Invalid option : "*option*"
	[Explanation]	"*option*" is not supported.
E0572200	[Message]	Illegal object file : "*file*"
	[Explanation]	The input file is not in the ELF format.
E0572500	[Message]	Cannot find library file : "*file*"
	[Explanation]	"*file*" specified as the library file was not found.
E0572501	[Message]	"*instance*" has been referenced as both an explicit specialization and a generated instantiation
	[Action by User]	For the file using "*instance*", confirm that form=relocate has not been used to generate a relocatable object file.
E0572502	[Message]	"*instance*" assigned to "*file1*" and "*file2*"
	[Explanation]	The definition of "*instance*" is duplicated in "*file1*" and "*file2*".
	[Action by User]	For the file using "*instance*", confirm that form=relocate has not been used to generate a relocatable object file.
E0573005	[Message]	Instantiation loop
	[Explanation]	An input file name may coincide with another file.
	[Action by User]	Change the filenames so that they do not coincide without the extension.

E0573007	[Message]	Cannot create instantiation request file "*file*"
	[Explanation]	The intermediate file for instantiation was not created.
	[Action by User]	Check whether access rights for the object creation directory are correct.
E0573008	[Message]	Cannot change to directory "*folder*"
	[Action by User]	Check if "*folder*" exists.
E0573009	[Message]	File "*file*" is read-only
	[Action by User]	Change the access rights.
E0573300	[Message]	Cannot open file : "*file*"
	[Action by User]	Check the filename and access rights.
E0573303	[Message]	Cannot read file : "*file*"
	[Explanation]	The input file was blank or there was not enough disk space.
E0573310	[Message]	Cannot open temporary file
	[Explanation]	An intermediate file cannot be opened. The HLNK_TMP specification was incorrect or there was not enough disk space.
E0573320	[Message]	Memory overflow
	[Explanation]	There is no more space in the usable memory within the linkage editor.
	[Action by User]	Increase the amount of memory available.
E0592001	[Message]	Multiple input files are not allowed.
	[Action by User]	Use a list file to convert more than one file.
E0592002	[Message]	Multiple output files are not allowed.
	[Action by User]	Use a list file to convert more than one file.
E0592003	[Message]	List file is specified more than once.
	[Action by User]	Combine them into a single list file.
E0592004	[Message]	Invalid argument for the "*option*" option.
	[Action by User]	Check the argument.
E0592005	[Message]	The "*option*" option can not have an argument.
	[Explanation]	An invalid argument was specified for the "*option*" option.
E0592006	[Message]	The "*option*" option requires an argument.
	[Explanation]	A required argument is missing from the "*option*" option specification.
E0592007	[Message]	The "*option*" option is specified more than once.
	[Explanation]	Option "*option*" can only be specified once at a time.
E0592008	[Message]	Requires an output file.
	[Explanation]	No output file has been specified for the specified input file.
E0592010	[Message]	Failed to open an output file "*file*".
E0592013	[Message]	Failed to delete a temporary file "*file*".
E0592015	[Message]	Failed to close an input file "*file*".
E0592016	[Message]	Failed to write an output file "*file*".

E0592018	[Message]	Failed to open an list file "*file*".
	[Action by User]	Make sure that the list file exists and has been specified correctly.
E0592019	[Message]	Syntax error in list file "*file*".
	[Explanation]	There is a syntax error in list file "*file*".
E0592020	[Message]	Failed to read a list file "*file*".
E0592101	[Message]	Unknown character '*C*'.
	[Explanation]	The pre-conversion C-language source file could not be converted, because it contains a character that is not permitted by the C language.
	[Action by User]	Edit the C-language source file and correct any syntax errors.
E0592102	[Message]	Illegal syntax in *string*.
	[Explanation]	The pre-conversion C-language source file could not be converted, because it contains a syntax error.
	[Action by User]	Edit the C-language source file and correct any syntax errors.
E0592201	[Message]	Illegal syntax.
	[Explanation]	The pre-conversion assembly source file could not be converted, because it contains a syntax error.
	[Action by User]	Edit the assembly source file and correct any syntax errors.
E0593002	[Message]	"-Xsfg_size_tidata_byte" size larger than "-Xsfg_size_tidata" size.
	[Action by User]	Set size "-Xsfg_size_tidata_byte" to equal to or less than size "-Xsfg_size_tidata", or size "-Xsfg_size_tidata" to greater than or equal to size "-Xsfg_size_tidata_byte".
E0593003	[Message]	Can not Read Symbol Information.
	[Explanation]	The symbol could not be loaded because there is no symbol-analysis information in memory, or it is corrupt.
	[Action by User]	Check the CX options and rebuild.
E0593004	[Message]	Can not Write the SFG file.
	[Explanation]	There could be a problem with disk space or user privileges.
	[Action by User]	Make sure that there is enough space to write the data, and check the user privileges.
E0594000	[Message]	Cannot find project file(*file name*).
	[Explanation]	There is no project file.
	[Action by User]	Make sure that the file exists.
E0594001	[Message]	Project file read error(*file name*).
	[Explanation]	An error occurred while loading the project file. Reading the project file may be blocked.
E0594002	[Message]	Illegal format in project file(*file name*).
	[Explanation]	The project file format is invalid.
	[Action by User]	This error occurs when invalid format is found in the project file. Either correct the error, or create the project again.

10.5.3 Abort errors

Table 10.4 Abort Errors

F0511128	[Message]	Library file "*file-name*" is not found.
F0512003	[Message]	Toomany 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.
F0520016	[Message]	"*file*" is not a valid source file name.
F0520035	[Message]	#error directive: *character string*
	[Explanation]	There is an "#error" directive in the source file.
F0520143	[Message]	Program too large or complicated to compile.
F0520163	[Message]	Could not open temporary file *xxx*.
F0520164	[Message]	Name of directory for temporary files is too long (*xxx*).
F0520182	[Message]	Could not open source file *xxx* (no directories in search list).
F0520189	[Message]	Error while writing "*file*" file.
F0520190	[Message]	Invalid intermediate language file.
F0520219	[Message]	Error while deleting file "*file*".
F0520542	[Message]	Could not create instantiation request file *name.*
F0520563	[Message]	Invalid preprocessor output file.
F0520564	[Message]	Cannot open preprocessor output file.
F0520583	[Message]	Invalid error output file.
F0520584	[Message]	Cannot open error output file.
F0520641	[Message]	*xxx* is not a valid directory.
F0520642	[Message]	Cannot build temporary file name.
F0520869	[Message]	Could not set locale *xxx* to allow processing of multibyte characters.
F0520919	[Message]	Invalid output file: *xxx*
F0520920	[Message]	Cannot open output file: *xxx*
F0520926	[Message]	Cannot open definition list file: *xxx*
F0521083	[Message]	Exported template file *xxx* is corrupted.
F0521151	[Message]	Mangled name is too long.
F0521335	[Message]	Cannot open predefined macro file: *xxx*
F0521336	[Message]	Invalid predefined macro entry at line *line*: *line2*
F0521337	[Message]	Invalid macro mode name *xxx*.

F0521338	[Message]	Incompatible redefinition of predefined macro *xxx*.
F0523029	[Message]	Cannot open rule file
	[Explanation]	The file specified in the -Xmisra2004="*file name*" option cannot be opened.
F0523030	[Message]	Incorrect description "*file name*" in rule file
	[Explanation]	The file specified in the -Xmisra2004="*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]	Illegal intrinsic function
	[Explanation]	An unsupported intrinsic function is called.
	[Action by User]	Check the -isa option specification.
F0523300	[Message]	Cannot open internal file.
	[Explanation]	An intermediate file internally generated by the compiler cannot be opened.
F0523301	[Message]	Cannot close internal file.
	[Explanation]	An intermediate file internally generated by the compiler cannot be closed.
F0523302	[Message]	Cannot write internal file.
	[Explanation]	An error occurred while an intermediate file was being written to.
F0530320	[Message]	Duplicate symbol "*symbol name*".
F0530800	[Message]	Type of symbol "*symbol-name*" differs between files.
F0530808	[Message]	Alignment of variable "*variable-name*" differs between files.
F0530810	[Message]	#pragma directive for symbol "*symbol-name*" differs between files.
F0531003	[Message]	The function "*function*" specified by the "*option*" option is not exist.
F0533015	[Message]	Symbol table overflow.
	[Explanation]	The number of symbols generated by the compiler exceeded the limit.
F0533021	[Message]	Out of memory.
	[Explanation]	Memory is insufficient.
	[Action by User]	Close other applications and recompile the program.
F0533300	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.

F0533301	[Message]	Cannot close an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be closed.
F0533302	[Message]	Cannot read an intermediate file.
	[Explanation]	An error occurred during reading of a temporary file.
F0533303	[Message]	Cannot write to an intermediate file.
	[Explanation]	An error occurred during writing of a temporary file.
F0533306	[Message]	Compilation was interrupted.
	[Explanation]	During compilation, an interrupt due to entry of the Cntl + C key combination was detected.
F0533330	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.
F0540027	[Message]	Cannot read file "*file-name*".
F0540204	[Message]	Illegal stack access.
	[Explanation]	Attempted usage of the stack by a function has exceeded 2 Gbytes.
	[Message]	[CC-RH] Illegal stack access.
	[Explanation]	[CC-RH] Attempted usage of the stack by a function has exceeded 64K bytes.
F0540300	[Message]	Cannot open an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be opened.
F0540301	[Message]	Cannot close an intermediate file.
	[Explanation]	A temporary file that was internally generated by the compiler cannot be closed.
F0540302	[Message]	Cannot read an intermediate file.
	[Explanation]	An error occurred during reading of a temporary file.
F0540303	[Message]	Cannot write to an intermediate file.
	[Explanation]	An error occurred during writing of a temporary file.
F0540400	[Message]	Different parameters are set for the same #pramga "*identifier*".
F0542001	[Message]	Cannot allocate register to "*symbol*" specified by the -Xr*number* option.
F0542002	[Message]	Cannot assign address constant to initializer for bitfield.
F0544302	[Message]	Cannot read an intermediate file.
	[Explanation]	An error occurred while an intermediate file was being read.
F0544802	[Message]	The value of the parameter for the in-line function is outside the defined range.
	[Explanation]	The value of the parameter for the inline function is outside the supported range.
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.
F0550515	[Message]	*string* not in .rept/.irp.
	[Action by User]	Check the directive.
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.
F0550517	[Message]	*string* nest over.
	[Explanation]	4294967294 or more levels of nesting have been used in *string*.
	[Action by User]	Check the nesting.
F0550518	[Message]	Unreasonable *macro_call* nesting.
	[Explanation]	A macro body called the macro currently being defined.
	[Action by User]	Check macro definition.

F0550519	[Message]	Argument mismatch.
	[Explanation]	The arguments specified in the macro call are invalid.
	[Action by User]	Check the arguments.
F0550520	[Message]	$ must be followed by defined symbol.
	[Explanation]	The dollar sign ($) must be followed by a non-symbol identifier or an undefined symbol name.
	[Action by User]	Check the code.
F0550521	[Message]	Too many errors.
	[Explanation]	The number of fatal errors has reached 30. Assembly halted.
	[Action by User]	Correct the errors.
F0550523	[Message]	Duplicated cpu type.
	[Explanation]	The target device has been specified more than once via options and/or directives.
	[Action by User]	Check the target-device specification.
F0550524	[Message]	Cannot find devicefile.
	[Explanation]	There is no device file corresponding to the specified target device, the device specification is incorrect, or no device is specified.
	[Action by User]	Check the device file.
F0550525	[Message]	Illegal microcontrollers.
	[Explanation]	There is no microcontrollers corresponding to the specified device file.
	[Action by User]	Check the device file.
F0550526	[Message]	Devicefile version mismatch, cannot use version *version*.
	[Action by User]	Check the device file.
F0550528	[Message]	tidata.byte/tibss.byte size overflow(size > 128).
	[Action by User]	Check the sizes of the .tidata.byte section and .tibss.byte section.
F0550529	[Message]	*section-name1*/*section-name2* size overflow(size > 256).
	[Action by User]	Check the sizes of the *section-name1* section and *section-name2* section, and .tibss section.
F0550530	[Message]	tidata/tibss size overflow(size > 256).
	[Action by User]	Check the sizes of the .tidata.byte section, .tibss.byte section, .tidata.word section, .tibss.word section, .tidata section, and .tibss section.
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.
F0550533	[Message]	Illegal size of function '*function*', cannot use '*function*' for $SMART_CORRECT.
	[Action by User]	Check the size of function "*function*".

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.
F0550535	[Message]	'*function1*' and '*function2*' are defined in same section.
	[Action by User]	Put a corrected function for which a "$smart_correct" control instruction has been specified in a different section from that of the uncorrected function.
F0550536	[Message]	"*Option*" option is not supported for this device.
	[Action by User]	Check the target device.
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 table 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.
F0550703	[Message]	Too many errors.
F0551601	[Message]	Illegal device information specified by "*source*".
F0551602	[Message]	Symbol is too long.
F0551604	[Message]	-mirror_source=1 option is not allowed for RL78-S1 core.
F0551605	[Message]	-mirror_region option is not allowed when -dev option is specified.
F0551606	[Message]	-mirror_region option is not allowed when -mirror_source=common option is specified.
F0551607	[Message]	Invalid value is specified as MIRROR area.
F0551608	[Message]	Specify addresses.
F0551609	[Message]	Unreasonable include file nesting.
	[Explanation]	The nesting level of the include is too deep or the function is recursively including itself.
	[Action by User]	Review the include file.

F0551610	[Message]	Unreasonable macro nesting.
	[Explanation]	The nesting level of the macro call is too deep or the function is recursively calling itself.
	[Action by User]	Review the macro definition.
F0553000	[Message]	Can't create file '*file name*'.
	[Explanation]	The *file name* file cannot be generated.
	[Action by User]	Check the directory capacity.
F0553001	[Message]	Can't open file '*file name*'.
	[Explanation]	The *file name* file cannot be opened.
	[Action by User]	Check the file name.
F0553002	[Message]	Can't write file '*file name*'.
	[Explanation]	The *file name* file cannot be written to.
	[Action by User]	Check the permission of the file.
F0553003	[Message]	Can't read file '*file name*'.
	[Explanation]	The *file name* file cannot be read.
	[Action by User]	Check the permission of the file.
F0553004	[Message]	Can't create Temporary file.
	[Explanation]	Temporary file cannot be generated.
	[Action by User]	Specify a directory in environment variable TMP_RX so that a temporary file will be created in some place other than the current directory.
F0553005	[Message]	Can't open Temporary file.
	[Explanation]	The temporary file cannot be opened.
	[Action by User]	Check the directory specified in TMP_RX.
F0553006	[Message]	Can't read Temporary file.
	[Explanation]	The temporary file cannot be read.
	[Action by User]	Check the directory specified in TMP_RX.
F0553007	[Message]	Can't write Temporary file.
	[Explanation]	The temporary file cannot be written to.
	[Action by User]	Check the directory specified in TMP_RX.
F0553008	[Message]	Illegal file name '*file name*'.
	[Explanation]	The file name is illegal.
	[Action by User]	Specify a file name that conforms to file name description rules.
F0553016	[Message]	Lacking cpu specification.
	[Explanation]	No CPU type is specified.
	[Action by User]	Specify the CPU type by the cpuoption or environment variable CPU_RX.

F0553100	[Message]	Command line is too long.
	[Explanation]	The command line has too many characters.
	[Action by User]	Re-input the command.
F0553101	[Message]	Invalid option '*xx*' is used.
	[Explanation]	An invalid command option *xx* is used.
	[Action by User]	The specified option is nonexistent. Re-input the command correctly.
F0553102	[Message]	Ignore option '*xx*'.
	[Explanation]	An invalid option is specified.
F0553103	[Message]	Option '*xx*' is not appropriate.
	[Explanation]	Command option *xx* is written incorrectly.
	[Action by User]	Specify the command option correctly again.
F0553104	[Message]	No input files specified.
	[Explanation]	No input file is specified.
	[Action by User]	Specify an input file.
F0553105	[Message]	Source files number exceed 80.
	[Explanation]	The number of source files exceeds 80.
	[Action by User]	Execute assembling separately in two or more operations.
F0553106	[Message]	Lacking cpu specification.
	[Explanation]	No CPU type is specified.
	[Action by User]	Specify the CPU type by the cpuoption or environment variable CPU_RX.
F0553110	[Message]	Multiple register base/fint_register.
	[Explanation]	A single register is specified by the baseand fint_registeroptions.
F0553111	[Message]	Multiple register base/pid.
	[Explanation]	A single register is specified by the baseand pidoptions.
F0553112	[Message]	Multiple register base/nouse_pid_register.
	[Explanation]	A single register is specified by the baseand nouse_pid_registeroptions.
F0553113	[Message]	Neither isa nor cpu is specified
F0553114	[Message]	Both '-isa' option and '-cpu' option are specified
F0553115	[Message]	The '-cpu' option and the '-fpu' option are inconsistent
F0553200	[Message]	Error occurred in executing '*xxx*'.
	[Explanation]	An error occurred when executing *xxx*.
	[Action by User]	Rerun asrx.
F0553201	[Message]	Not enough memory.
	[Explanation]	Memory is insufficient.
	[Action by User]	Divide the file and re-run. Orincrease the memory capacity.

F0553202	[Message]	Can't find work dir.
	[Explanation]	The work directory is not found.
	[Action by User]	Make sure that the setting of environment variable TMP_RX is correct.
F0560001	[Message]	Can not open input file "*file*"
F0560002	[Message]	Can not open output file "*file*"
F0560003	[Message]	Can not create output file "*file*"
F0560004	[Message]	Can not open directive file "*file*"
F0560005	[Message]	Can not get size of directive file "*file*"
F0560006	[Message]	Can not truncate output file "*file*" to have size(*number*)
F0560007	[Message]	Can not seek output file "*file*"
F0560008	[Message]	Can not write output file "*file*"
F0560009	[Message]	Can not find devicefile "*string*"
F0560010	[Message]	Illegal devicefile "*string*"
F0560011	[Message]	Can not open devicefile "*string*"
F0560012	[Message]	Can not read devicefile "*string*"
F0560013	[Message]	Illegal object file(Error number:*number*)
F0560101	[Message]	Illegal ELF version
F0560102	[Message]	Illegal target machine type
F0560103	[Message]	Illegal target machine class
F0560104	[Message]	Illegal target machine byte order
F0560105	[Message]	Illegal ELF file type, must be relocatable or shared library file
F0560106	[Message]	Unknown format type file "*file*"
F0560107	[Message]	Illegal devicefile. different family "*string*".
F0560108	[Message]	"*file*"is not executable file.
F0560109	[Message]	"*file*"is rom packed file.
F0560110	[Message]	Multiple inclusion of same file attempted, ignored.
F0560112	[Message]	Can not link *string1* common objects with *string2* objects. "*file*" is *string1* common object.
F0560113	[Message]	Can not link mask reg using objects with mask reg not using objects. "*file*" is mask reg using object.
F0560114	[Message]	Input files have different BPC value.
F0560115	[Message]	Can not link using objects with the specified device.
F0560201	[Message]	Fail to get symbol name string.
F0560202	[Message]	Fail to get *number* th symbol name string
F0560203	[Message]	Symbol "*symbol*" has unknown binding class(*number*)
F0560204	[Message]	Weak symbol "*symbol*" not supported.
F0560208	[Message]	Symbol "*symbol*" multiply defined.

F0560209	[Message]	Linking of symbol "*symbol*" in sdata or sbss attribute section in "*file1*" and in order attribute section in "*file2*" is attempted.
F0560210	[Message]	Can not find entry point symbol "*symbol*" specified with -Xentry_address option.
F0560213	[Message]	"$EXT_FUNC *symbol1*, *ID1*" is already defined as "$EXT_FUNC *symbol2*, *ID2*" in other file.
F0560215	[Message]	Too many symbols.
F0560216	[Message]	Multiple defined symbol. symbol defined previous defined file *symbol* *file1* *file2*
F0560217	[Message]	Undefined symbol. "*symbol*" referenced in "*file*".
F0560218	[Message]	Symbol table overflow.
F0560219	[Message]	"*symbol*" has different $EXT_ENT_SIZE.
F0560220	[Message]	TP-symbol"*symbol*" specified as GP-symbol"*symbo2*"'s base symbol is not found.
F0560301	[Message]	Failed to get section name string table section.
F0560302	[Message]	Failed to get symbol name string table section.
F0560303	[Message]	Failed to get section header.
F0560304	[Message]	Failed to get section name string.
F0560306	[Message]	Section "*section*" has unknown section type (*number*).
F0560307	[Message]	Can not get raw data of section "*section*".
F0560309	[Message]	Interrupt function section "*section*" is already defined.
F0560310	[Message]	Special section "*secton*" is already defined.
F0560311	[Message]	Illegal "*section*" section size.
F0560404	[Message]	Relocated value(*value*) of relocation entry(symbol:*symbol*, file:*file*, section:*section*, offset:*offset*, type:*relocation type*) for branch command become odd value.
F0560405	[Message]	Can not find GP-symbol in segment "segment" of illegal label reference for local symbol in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*).local symbol is allocated in section "*section1*" (file:*file1*).
F0560407	[Message]	Can not find GP-symbol in segment "*segment*" of illegal label reference for symbol "*symbol*" in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file1*).
F0560409	[Message]	Can not find GP-symbol in section "*section*" of file "*file1*" or illegal label reference for symbol "*symbol*" in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file1*).
F0560411	[Message]	Relocated value(*value*) of relocation entry(file:*file*, section:*section*, offset:*offset*, type:*relocation type*) for branch command become odd value.
F0560414	[Message]	EP symbol is needed for using SIDARA/SEDATA segment.
F0560415	[Message]	Symbol "*symbol*" (output section *section1*)is too far from output section "*section2*". (value:*value*, file:*file* input section:*section3*, offset;*offset*, type:*relocation type*)
F0560417	[Message]	Output section "*section1*" is too far from output section "*section2*". (value:*value*, file:*file* input section:*section3*, offset;*offset*, type:*relocation type*)
F0560419	[Message]	Output section "*section1*" overflowed or illegal label reference for symbol "*symbol*" in file "*file*". (value:*value*, input section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file*)

F0560421	[Message]	Output section "*section1*" overflowed or illegal label reference for local symbol in file "*file2*". (value:*value*, input section:*section2*, offset;*offset*, type:*relocation type*). local symbol is allocated in section "*section1*" (file:*file1*)
F0560423	[Message]	Can not reference extern symbol "*symbol*" by string.
F0560424	[Message]	No symbol information in boot file "*file*"
F0560502	[Message]	Can not find library member at offset(offset) specified in library symbol table entry.
F0560503	[Message]	Sorry. shared library not supported.
F0560627	[Message]	Segment "*segment*" overflowed highest address of target machine.
F0560629	[Message]	Segment directive of segment "*segment*" needs string.
F0560630	[Message]	Section directive of section "*section*" needs string.
F0560631	[Message]	Symbol directive of symbol "*symbol*" needs string.
F0560633	[Message]	Start address(*number1*) of segment "*segment1*" overlaps previous segment "*segment2*" ended before address(*number2*).
F0560634	[Message]	Start address(*number1*) of section "*section1*" overlaps previous section "*section2*" ended before address(*number2*).
F0560635	[Message]	Start address(*number1*) of section "*section*" overflowed start address(*number2*) of segment "*segment*".
F0560636	[Message]	Memory size(*number1*) of segment "*segment*" overflowed specified or default maximum memory size(*number2*).
F0560649	[Message]	End address of section "*section*" overflowed maximum memory address(*number*).
F0560650	[Message]	End address of segment "*segment*" overflowed maximum memory address(*number*).
F0560652	[Message]	Segment "*segment*"(*number1-number2*) overflowed highest or lowest address of internal memory (*number3-number4*).
F0560657	[Message]	Section "*section*" must be specified in link directive.
F0560658	[Message]	Description of section "*section*" in mapping directive is illegal.
F0560661	[Message]	Can not allocate memory(builtin new error).
F0560662	[Message]	CallTBasePointer(CTBP) is not set. CTBP must be set when compiler option "-Ospeed" (or "-Xpro_epi_runtime=off")is not specified.
F0560701	[Message]	"*string1*" option is illegal when "*string2*" option is specified.
F0560705	[Message]	Unknown cpu type.
F0560707	[Message]	"$ext_func" is illegal when " -Xrelinkable_object" option is specified.
F0560708	[Message]	"*string1*" option needs "string2" option.
F0560712	[Message]	"*string*" option's value overflowed.
F0561001	[Message]	"*file*" bad magic.
F0561002	[Message]	*file*: not absolute object.
F0561003	[Message]	Illegal input file type. file"*file*" is library file.
F0561004	[Message]	Memory allocation fault.
F0561005	[Message]	Address of symbol "*symbol* " must be same in all files.
F0561006	[Message]	*file*: symbol"*symbol*" not found.

F0561007	[Message]	rompsec section overflowed highest address of target machine.
F0561008	[Message]	Section "*section*" and "*section2*" section overlapped.
F0561009	[Message]	Symbol(*start_label*) must be word alignment.
F0561010	[Message]	Packing section not found.
F0561011	[Message]	Section "*section*" must be word aligned in internal instruction RAM.
F0561012	[Message]	Section not found.
F0561013	[Message]	*file*: illegal section type "section" specified with -Xrompsec_data option.
F0561014	[Message]	*file*: illegal section type "section" specified with -Xrompsec_text option.
F0561015	[Message]	-Xrompsec_data option section "*section*" not found.
F0561016	[Message]	-Xrompsec_text option section "*section*" not found.
F0561019	[Message]	Processor type must be same in all files.
F0562001	[Message]	No section data exists in specified address area (*address1-address2*).
F0562002	[Message]	Can not get section "*section*".
F0562003	[Message]	"*file*" is not ELF file.
F0562004	[Message]	"*file*" is library file.
F0562005	[Message]	Illegal object file (*string*).
F0562006	[Message]	File name "*name*" is too long.
F0562007	[Message]	Can not create HEX rom data, because there is no memory information.
F0562008	[Message]	*section* : no such section.
F0562009	[Message]	Memory allocation fault.
F0562014	[Message]	Section "*section*" overflowed lowest address of internal memory.
F0562028	[Message]	CRC *xxx* option's parameter error (*xxx*).
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.
F0563004	[Message]	Unsupported inter-module optimization information type "*type*" in "*file*"
	[Explanation]	The file contains an unsupported inter-module optimization information *type*.
	[Action by User]	Check if the compiler and assembler versions are correct.
F0563010	[Message]	No mirror region information
	[Explanation]	The allocation address information of the mirror region is not specified.
	[Action by User]	Check if the -far_rom, -mirror_region, or -dev option is correct

F0563020	[Message]	No cpu information in input files
	[Explanation]	The CPU type cannot be identified from the input file.
	[Action by User]	Check the input file.
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*"
	[Explanation]	Data addresses overlap within an absolute address section.
	[Action by User]	Modify the source program.
F0563110	[Message]	Illegal cpu type "*cpu type*" in "*file*"
	[Explanation]	A file with a different cpu type was input.
F0563111	[Message]	Illegal encode type "*endian type*" in "*file*"
	[Explanation]	A file with a different endian type was input.
F0563112	[Message]	Invalid relocation type in "*file*"
	[Explanation]	There is an unsupported relocation type in *file*.
	[Action by User]	Ensure the compiler and assembler versions are correct.
F0563113	[Message]	Illegal mode type "*mode*" in "*file*"
	[Explanation]	A "*mode*" file that cannot be mixed is input.
	[Action by User]	Check if the compiler and assembler options and the device file are correct.
F0563114	[Message]	Illegal cpu type "*CPU type*" in device file "*file*"
	[Explanation]	"*CPU type*" is different.
	[Action by User]	Check if the device file is correct.
F0563120	[Message]	Illegal size of the absolute code section : "*section*" in "*file*"
	[Explanation]	Absolute-addressing section *section* in *file* has an illegal size.
F0563121	[Message]	Illegal type of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563122	[Message]	Illegal attribute of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563123	[Message]	Gap is within the limits of the section : "*section*"
	[Explanation]	"*section*" cannot be allocated.
	[Action by User]	Check the source file.
F0563124	[Message]	Illegal alignment of the section : "*section*"
	[Explanation]	"*section*" cannot be allocated by section alignment.
	[Action by User]	Check the source file.

F0563125	[Message]	Illegal kind of the section : "*section*" in "*file*"
	[Explanation]	"*section*" type is different.
	[Action by User]	Check the section specification of a source file.
F0563130	[Message]	Range "*range*" in "*file*" conflicts with that in another file
	[Explanation]	A file which has a different memory area for "*range*" is input.
	[Action by User]	Check if the compiler and assembler options are correct.
F0563140	[Message]	No "*area*" area information in input device file
	[Explanation]	The device file does not contain the "*area*" information.
	[Action by User]	Check if the device file is correct.
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.
F0563311	[Message]	Cannot close temporary file
	[Explanation]	A temporary file cannot be closed. There may be insufficient disk space.
F0563312	[Message]	Cannot write temporary file
	[Explanation]	Writing to a temporary file is not possible. There may be insufficient disk space.
F0563313	[Message]	Cannot read temporary file
	[Explanation]	A temporary file cannot be read. 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.
F0563400	[Message]	Cannot execute "*load module*"
	[Explanation]	*load module* cannot be executed.
	[Action by User]	Check whether the path for *load module* is set correctly.
F0563410	[Message]	Interrupt by user
	[Explanation]	An interrupt generated by (Ctrl) + C keys from a standard input terminal was detected.
F0563420	[Message]	Error occurred in "*load module*"
	[Explanation]	An error occurred while executing *load module*.
F0563430	[Message]	The total section size exceeded the limit
	[Explanation]	The limit on the amount of object code that can be linked by the evaluation version was exceeded. [RH850] The limit on the amount of object code to be linked is 256 Kbytes. [RX] The limit on the amount of object code to be linked is 128 Kbytes. [RL78] The limit on the amount of object code to be linked is 64 Kbytes.
	[Action by User]	Ensure that the amount of object code to be linked does not exceed the limit or purchase the product version.
F0578200	[Message]	memory allocation fault
	[Explanation]	Not enough memory.
F0578201	[Message]	bad key *character* - use [dm(a|b)qr(a|b|u)txV]
	[Explanation]	*character* cannot be specified as a key.
F0578202	[Message]	bad option *character* - use [cv]
	[Explanation]	*character* cannot be specified as an option.
F0578203	[Message]	bad option *string*
	[Explanation]	*string* cannot be specified as an option.

F0578204	[Message]	can not create file *file*
	[Explanation]	Could not create file *file*.
F0578205	[Message]	file name *name*... is too long - limit is *number*
	[Explanation]	File name *name* is too long. The maximum value is *number1*.
F0578206	[Message]	can not open file *file*
	[Explanation]	Could not open file *file*.
F0578207	[Message]	can not close file *file*
	[Explanation]	Could not close file *file*.
F0578208	[Message]	can not read file *file*
	[Explanation]	Cannot read from file *file*.
F0578209	[Message]	can not write file *file*
	[Explanation]	Cannot write to file *file*.
F0578210	[Message]	can not seek file *file*
	[Explanation]	Cannot seek in file *file*.
F0578212	[Message]	can not nest command file *file*
	[Explanation]	Command file *file* is nested. Nesting is not allowed.
F0578213	[Message]	file is not library *file*
	[Explanation]	*file* is not a library file.
F0578214	[Message]	malformed library file *file*
	[Explanation]	Library file *file* could be corrupt.
F0578215	[Message]	can not find member *member*
	[Explanation]	Member *member* not found in library file.
F0578216	[Message]	symbol table limit error *file* (*number1*) - limit is *number2*
	[Explanation]	The number of symbols *number1* in library file *file* exceeds the maximum limit. The maximum value is *number2*.
F0578217	[Message]	symbol table error *file*
	[Explanation]	Failed to create a library file table for library file *file*.
F0578218	[Message]	string table error *file*
	[Explanation]	The library string table for library file *file* could be corrupt.
F0578219	[Message]	*file* has no member
	[Explanation]	There are no members in library file *file*.
F0578220	[Message]	version error *file*
	[Explanation]	The version of the format of the specified file *file* is not supported by this librarian.
F0578221	[Message]	can not read library header *file*
	[Explanation]	Cannot read header from library file *file*.
F0593113	[Message]	Neither isa nor cpu is specified
F0593114	[Message]	Both '-isa' option and '-cpu' option are specified

F0595001	[Message]	Cannot open file "*file name*".
F0595002	[Message]	Cannot read file "*file name*".
F0595003	[Message]	Cannot write file "*file name*".
F0595004	[Message]	Cannot close file "*file name*".

10.5.4 Information

Table 10.5 Informations

M0520009	[Message]	Nested comment is not allowed.
M0520018	[Message]	Expected a ")".
M0520111	[Message]	Statement is unreachable.
M0520128	[Message]	Loop is not reachable from preceding code.
M0520174	[Message]	Expression has no effect.
M0520193	[Message]	Zero used for undefined preprocessing identifier *xxx*.
M0520237	[Message]	Selector expression is constant.
M0520261	[Message]	Access control not specified ("*name*" by default).
M0520324	[Message]	Duplicate friend declaration.
M0520381	[Message]	Extra ";" ignored.
M0520399	[Message]	*name* has an operator new *xxx*() but no default operator delete *xxx*().
M0520400	[Message]	*name* has a default operator delete *xxx*() but no operator new *xxx*().
M0520479	[Message]	*name* redeclared "inline" after being called.
M0520487	[Message]	Inline *name* cannot be explicitly instantiated.
M0520534	[Message]	Use of a local type to specify an exception.
M0520535	[Message]	Redundant type in exception specification.
M0520549	[Message]	*symbol* is used before its value is set.
M0520618	[Message]	Struct or union declares no named members.
M0520652	[Message]	Calling convention is ignored for this type.
M0520678	[Message]	Call of "*symbol*" cannot be inlined.
M0520679	[Message]	*symbol* cannot be inlined.
M0520815	[Message]	Type qualifier on return type is meaningless.
M0520831	[Message]	Support for placement delete is disabled.
M0520863	[Message]	Effect of this "#pragma pack" directive is local to *xxx*.
M0520866	[Message]	Exception specification ignored.
M0520949	[Message]	Specifying a default argument on this declaration is nonstandard.
M0521348	[Message]	Declaration hides "*symbol*".
M0521353	[Message]	*symbol* has no corresponding member operator delete *xxx* (to be called if an exception is thrown during initialization of an allocated object).
M0521380	[Message]	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
M0521381	[Message]	Carriage return character in source line outside of comment or character/string literal.
M0523009	[Message]	This pragma has no effect.

M0523028	[Message]	Rule *rule number* : *description*
	[Explanation]	The indicated location of the *rule number* of MISRA-C:2004 and the *description* was detected.
M0523033	[Message]	Precision lost
	[Explanation]	There is a possibility that precision will be lost when converting the value of the expression on the right side to the type on the left side in an assignment expression.
M0560001	[Message]	Section "*section*" created by optimization "*optimization*"
	[Explanation]	The section named *section* was created as a result of the optimization.
M0560002	[Message]	Symbol "*symbol*" created by optimization "*optimization*"
	[Explanation]	The symbol named *symbol* was created as a result of the optimization.
M0560004	[Message]	"*file*"-"*symbol*" deleted by optimization
	[Explanation]	As a result of symbol_delete optimization, the symbol named *symbol* in *file* was deleted.
M0560005	[Message]	The offset value from the symbol location has been changed by optimization : "*file*"-"*section*"-"*symbol* ± offset"
	[Explanation]	As a result of the size being changed by optimization within the range of symbol ± offset, the offset value was changed. Check that this does not cause a problem. To disable changing of the offset value, cancel the specification of the goptimize option on assembly of file.
M0560100	[Message]	No inter-module optimization information in "*file*"
	[Explanation]	No inter-module optimization information was found in *file*. Inter-module optimization is not performed on *file*. To perform inter-module optimization, specify the goptimize option on compiling and assembly.
M0560101	[Message]	No stack information in "*file*"
	[Explanation]	No stack information was found in *file*. *file* may be an assembler output file. The contents of the file will not be in the stack information file output by the optimizing linker.
M0560102	[Message]	Stack size "*size*" specified to the undefined symbol "*symbol*" in "*file*"
	[Explanation]	Stack size *size* is specified for the undefined symbol named symbol in *file*.
M0560103	[Message]	Multiple stack sizes specified to the symbol "*symbol*"
	[Explanation]	Multiple stack sizes are specified for the symbol named *symbol*.
M0560300	[Message]	Mode type "*mode type1*" in "*file*" differ from "*mde type2*"
	[Explanation]	A file with a different mode type was input.
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*.
M0560510	[Message]	Section "*section*" was moved other area specified in option "cpu=<*attribute*>"
	[Explanation]	*section* without dividing is allocated according to cpu=<*attribute*>.

M0560511	[Message]	Sections "*section name*","*new section name*" are Non-contiguous
	[Explanation]	*section* was divided and the newly created section is new section name.
M0560512	[Message]	Section "*section*" created by "*option*"
M0592150	[Message]	*string1* was converted into *string2*.
M0592151	[Message]	*string1* was deleted.
M0592152	[Message]	string was converted for CX compiler.
M0592153	[Message]	Bit access of I/O register wad converted into macro call.
M0592154	[Message]	Inserted macro definition for bit access of I/O register.
M0592155	[Message]	'multi' option was added to #pragma interrupt.
M0592156	[Message]	Inserted #pragma interrupt NO_VECT.
M0592157	[Message]	Inserted *string*.
M0592250	[Message]	'*string1*' has been changed to '*string2*'
M0592251	[Message]	Multi Statement has been changed to Newline.(;)
M0592252	[Message]	'*string*' has been changed to syntax of CX
M0592253	[Message]	'*string1*' has been changed to '*string2*'. Specify external symbol.(.extern)
	[Explanation]	'.option data' or '.option sdata' has been changed to '$data' or '$sdata'. CX requires symbols to be declared beforehand via .extern. Please make sure that the symbol is declared.
M0592270	[Message]	'*string*' has been deleted.
M0592280	[Message]	'.previous' cannot be changed. Specify appropriate section.
	[Explanation]	".previous" are not supported by CX.
M0592281	[Message]	'*string*' cannot be changed.(bit:value) Specify appropriate value by 'string'
	[Explanation]	string 'bit:value' are not supported by CX.
M0592282	[Message]	'.option reg_mode' cannot be changed. Specify appropriate register mode.
	[Explanation]	Values specified with ".option reg_mode" are not supported by CX.
M0594201	[Message]	Do you want to overwrite it?
	[Action by User]	This will overwrite the existing file. Click [Yes] to save and overwrite. Click [No] to cancel.
M0594202	[Message]	Do you want to cancel adjustments to the selected function(*function name*)?
	[Action by User]	Return the function's {function name} setting to its initial value. Click [Yes] to revert to the initial value. Click [No] to cancel.
M0594203	[Message]	Do you want to cancel adjustments to all the functions?
	[Action by User]	Return all function settings to their initial values. Click [Yes] to revert to the initial values. Click [No] to cancel.

10.5.5 Warnings

Table 10.6 Warnings

W0511105	[Message]	"*path*" specified by the "*character string*" option is a file. Specify a folder.
W0511106	[Message]	The folder "*folder*" specified by the "*character string*" option is not found.
W0511123	[Message]	The "*character string2*" option is ignored when the "*character string1*" option is specified at the same time.
W0511143	[Message]	The "-Xfloat" option is ignored because specified device does not have FPU.
W0511144	[Message]	"-C" option and "-Xcommon" option is mismatch. Instruction set by "*character string1*" option is ignored. Create common object for "*character string2*" instruction set.
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.
W0511156	[Message]	Device file is not found in the folder specified by the "-Xdev_path" option.
	[Explanation]	Device file is not found in the folder specified by the "-Xdev_path" option. Will search standard device file folder.
W0511164	[Message]	[CC-RH] Dupulicate file name. "*file-name*".
	[Explanation]	[CC-RH] 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.
	[Message]	Dupulicate file name. "*file-name*".
W0511166	[Message]	"*macro name*" is not a valid predefined macro name.
	[Explanation]	Specification of the undefine option is invalid.
W0511168	[Message]	"*option-name*" option has no effect in this version.
W0511169	[Message]	"*code*" is not valid in "*language specifications*"
	[Explanation]	"*code*" is invalid in the "language specifications (C or C++)".
W0511170	[Message]	"*option-name*" option is ignored due to the specification of another option.
W0511171	[Message]	"*code*" is ignored in "*language specifications*".
	[Explanation]	"*code*" is ignored in the "language specifications (C or C++)".
W0511172	[Message]	Nothing to compile, assemble or link.(input and output combination)
	[Action by User]	Check the combination of the input file and output option.
W0519999	[Message]	The "*option-name*" option is not implemented.
W0520001	[Message]	Last line of file ends without a newline.
	[Action by User]	Add a line break.

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.
W0520014	[Message]	Extra text after expected end of preprocessing directive.
W0520019	[Message]	Extra text after expected end of number.
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*").
W0520045	[Message]	#undef may not be used on this predefined name.
W0520046	[Message]	*symbol* is predefined; attempted redefinition ignored.
W0520047	[Message]	Incompatible redefinition of macro "symbol".
W0520054	[Message]	Too few arguments in macro invocation.
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 in ANSI-C.
W0520063	[Message]	Shift count is too large.
W0520064	[Message]	Declaration does not declare anything.
W0520066	[Message]	Enumeration value is out of "int" range.
W0520068	[Message]	Integer conversion resulted in a change of sign.
W0520069	[Message]	Integer conversion resulted in truncation.
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.
W0520085	[Message]	Invalid storage class for a parameter.
W0520086	[Message]	Invalid storage class for a function.
W0520099	[Message]	A declaration here must declare a parameter.

W0520101	[Message]	*xxx* has already been declared in the current scope.
W0520108	[Message]	Signed bit field of length 1.
W0520111	[Message]	Statement is unreachable.
W0520117	[Message]	Non-void "*function name*" should return a value.
W0520118	[Message]	A void function may not 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.
W0520139	[Message]	Taking the address of a bit field is not allowed.
W0520140	[Message]	Too many arguments in function call.
W0520144	[Message]	A value of type "*type1*" cannot be used to initialize an entity of type "*type2*".
W0520147	[Message]	Declaration is incompatible with "*declaration*" (declared at line *number*).
W0520152	[Message]	Conversion of nonzero integer to pointer.
W0520157	[Message]	Expression must be an integral constant expression.
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.
W0520181	[Message]	Argument is incompatible with corresponding format string conversion.
W0520185	[Message]	Dynamic initialization in unreachable code.
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.
W0520223	[Message]	Function *xxx* declared implicitly.
W0520224	[Message]	The format string requires additional arguments.
W0520225	[Message]	The format string ends before this argument.

W0520226	[Message]	Invalid format string conversion.
W0520229	[Message]	Bit field cannot contain all values of the enumerated type.
W0520231	[Message]	Declaration is not visible outside of function.
W0520232	[Message]	Old-fashioned typedef of "void" ignored.
W0520236	[Message]	Controlling expression is constant.
W0520257	[Message]	const "*symbol*" requires an initializer.
W0520260	[Message]	Explicit type is missing ("int" assumed).
W0520262	[Message]	Not a class or struct name.
W0520280	[Message]	Declaration of a member with the same name as its class.
W0520284	[Message]	NULL reference is not allowed.
W0520296	[Message]	Invalid use of non-lvalue array.
W0520300	[Message]	A pointer to a bound function may only be used to call the function.
W0520301	[Message]	typedef name has already been declared (with same type).
W0520326	[Message]	Inline is not allowed.
W0520335	[Message]	Linkage specification is not allowed.
W0520368	[Message]	*xxx* defines no constructor to initialize the following:
W0520370	[Message]	*symbol* has an uninitialized const field.
W0520375	[Message]	Declaration requires a typedef name.
W0520377	[Message]	"virtual" is not allowed.
W0520381	[Message]	Extra ";" ignored.
W0520382	[Message]	In-class initializer for nonstatic member is nonstandard.
W0520414	[Message]	Delete of pointer to incomplete class.
W0520430	[Message]	Returning reference to local temporary.
W0520494	[Message]	Declaring a void parameter list with a typedef is nonstandard.
W0520497	[Message]	Declaration of %sq hides template parameter.
W0520512	[Message]	Type qualifier on a reference type is not allowed.
W0520513	[Message]	A value of type "*type1*" cannot be assigned to an entity of type "*type2*".
W0520514	[Message]	Pointless comparison of unsigned integer with a negative constant.
W0520520	[Message]	Initialization with "{...}" expected for aggregate object.
W0520522	[Message]	Pointless friend declaration.
W0520523	[Message]	"." used in place of "::" to form a qualified name.
W0520533	[Message]	Handler is potentially masked by previous handler for type "type".
W0520541	[Message]	Omission of exception specification is incompatible with previous *name*.
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.

W0520552	[Message]	Exception specification is not allowed.
W0520553	[Message]	external/internal linkage conflict for "symbol".
W0520554	[Message]	*name* will not be called for implicit or explicit conversions.
W0520611	[Message]	Overloaded virtual function *name1* is only partially overridden in *name2*.
W0520614	[Message]	Invalid error number in diagnostic control option:
W0520617	[Message]	Pointer-to-member-function cast to pointer to function.
W0520618	[Message]	struct or union declares no named members.
W0520650	[Message]	Calling convention specified here is ignored.
W0520657	[Message]	Inline specification is incompatible with previous "*symbol*".
W0520662	[Message]	Call of pure virtual function.
W0520676	[Message]	Using out-of-scope declaration of *type* "*symbol*" (declared at line *number*).
W0520691	[Message]	*xxx*, required for copy that was eliminated, is inaccessible.
W0520692	[Message]	*xxx*, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
W0520708	[Message]	Incrementing a bool value is deprecated.
W0520720	[Message]	Redeclaration of *xxx* is not allowed to alter its access.
W0520722	[Message]	Use of alternative token "<:" appears to be unintended.
W0520723	[Message]	Use of alternative token "%%:" appears to be unintended.
W0520737	[Message]	Using-declaration ignored -- it refers to the current namespace.
W0520748	[Message]	Calling convention specified more than once.
W0520760	[Message]	*symbol* explicitly instantiated more than once.
W0520767	[Message]	Conversion from pointer to smaller integer.
W0520780	[Message]	Reference is to *symbol1* -- under old for-init scoping rules it would have been *symbol2*.
W0520783	[Message]	Empty comment interpreted as token-pasting operator "##".
W0520794	[Message]	Template parameter %sq may not be used in an elaborated type specifier.
W0520802	[Message]	Specifying a default argument when redeclaring an unreferenced function template is nonstandard.
W0520806	[Message]	Omission of exception specification is incompatible with *name*.
W0520812	[Message]	const object requires an initializer -- class *type* has no explicitly declared default constructor.
W0520815	[Message]	Type qualifier on return type is meaningless.
W0520825	[Message]	Virtual inline *name* was never defined.
W0520826	[Message]	*name* was never referenced.
W0520829	[Message]	Double used for "long double" in generated C code.
W0520830	[Message]	*xxx* has no corresponding operator delete*yyy* (to be called if an exception is thrown during initialization of an allocated object).
W0520831	[Message]	Support for placement delete is disabled

W0520836	[Message]	Returning reference to local variable.
W0520837	[Message]	Omission of explicit type is nonstandard ("int" assumed).
W0520867	[Message]	Declaration of "size_t" does not match the expected type "*type*".
W0520870	[Message]	Invalid multibyte character sequence.
W0520902	[Message]	Type qualifier ignored.
W0520912	[Message]	Ambiguous class member reference -- *symbol1* used in preference to *symbol2*.
W0520925	[Message]	Type qualifiers on function types are ignored.
W0520936	[Message]	Redeclaration of *name* alters its access.
W0520940	[Message]	Missing return statement at end of non-void "*symbol*"..
W0520941	[Message]	Duplicate using-declaration of *name* ignored.
W0520942	[Message]	enum bit-fields are always unsigned, but enum %t includes negative enumerator.
W0520948	[Message]	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
W0520951	[Message]	Return type of function "main" must be "int".
W0520959	[Message]	Declared size for bit field is larger than the size of the bit field type; truncated to any-string bits.
W0520961	[Message]	Use of a type with no linkage to declare a variable with linkage.
W0520962	[Message]	Use of a type with no linkage to declare a function.
W0520970	[Message]	The qualifier on this friend declaration is ignored.
W0520973	[Message]	Inline used as a function qualifier is ignored.
W0520984	[Message]	operator new and operator delete cannot be given internal linkage.
W0520991	[Message]	Extra braces are nonstandard.
W0520993	[Message]	Subtraction of pointer types "*type name1*" and "*type name2*" is nonstandard.
W0520997	[Message]	*function2* is hidden by function*1* -- virtual function override intended?
W0521000	[Message]	A storage class may not be specified here.
W0521028	[Message]	Invalid redeclaration of nested class.
W0521030	[Message]	A variable with static storage duration cannot be defined within an inline function.
W0521046	[Message]	Floating-point value cannot be represented exactly.
W0521050	[Message]	Imaginary *= imaginary sets the left-hand operand to zero.
W0521051	[Message]	Standard requires that "*symbol*" be given a type by a subsequent declaration ("int" assumed).
W0521053	[Message]	Conversion from integer to smaller poinster.
W0521055	[Message]	Types cannot be declared in anonymous unions.
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.
W0521145	[Message]	*type1* would have been promoted to "*type2*" when passed through the ellipsis parameter; use the latter type instead.
W0521163	[Message]	va_start should only appear in a function with an ellipsis parameter.
W0521192	[Message]	Null (zero) character in input line ignored.
W0521193	[Message]	Null (zero) character in string or character constant.
W0521194	[Message]	Null (zero) character in header name.
W0521197	[Message]	The prototype declaration of %nfd is ignored after this unprototyped redeclaration.
W0521211	[Message]	Nonstandard cast to array type ignored.
W0521213	[Message]	field uses tail padding of a base class.
W0521218	[Message]	Base class *xxx* uses tail padding of base class *yyy*.
W0521222	[Message]	Invalid error number.
W0521223	[Message]	Invalid error tag.
W0521224	[Message]	Expected an error number or error tag.
W0521235	[Message]	Nonstandard conversion between pointer to function and pointer to data.
W0521273	[Message]	Alignment-of operator applied to incomplete type.
W0521285	[Message]	Nonstandard qualified name in namespace member declaration.
W0521290	[Message]	Non-POD class type passed through ellipsis.
W0521294	[Message]	Integer operand may cause fixed-point overflow.
W0521296	[Message]	Fixed-point value cannot be represented exactly.
W0521297	[Message]	Constant is too large for long long; given unsigned long long type (nonstandard).
W0521301	[Message]	*xxx* declares a non-template function -- add <> to refer to a template instance.
W0521302	[Message]	Operation may cause fixed-point overflow.
W0521307	[Message]	Class member typedef may not be redeclared.
W0521308	[Message]	Taking the address of a temporary.
W0521310	[Message]	Fixed-point value implicitly converted to floating-point type.
W0521316	[Message]	Value cannot be converted to fixed-point value exactly.
W0521319	[Message]	Fixed-point operation result is out of range.
W0521342	[Message]	const_cast to enum type is nonstandard.
W0521346	[Message]	Function returns incomplete class type %t.
W0521361	[Message]	Negation of an unsigned fixed-point value.
W0521373	[Message]	Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
W0521374	[Message]	Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
W0521375	[Message]	Conversion from pointer to same-sized integral type (potential portability problem).

W0521386	[Message]	Storage specifier ignored.
W0521396	[Message]	White space between backslash and newline in line splice ignored.
W0521400	[Message]	positional format specifier cannot be zero.
W0521420	[Message]	Some enumerator values cannot be represented by the integral type underlying the enum type.
W0521422	[Message]	Multicharacter character literal (potential portability problem).
W0521427	[Message]	offsetof applied to non-POD types is nonstandard.
W0521433	[Message]	No prior push_macro for *xxx*.
W0521443	[Message]	__real/__imag applied to real value.
W0521444	[Message]	*symbol* was declared "deprecated (*xxx*)".
W0521546	[Message]	Argument must be a constant null pointer value.
W0521547	[Message]	Insufficient number of arguments for sentinel value.
W0521548	[Message]	Sentinel argument must correspond to an ellipsis parameter.
W0521551	[Message]	No #pragma start_map_region is currently active: pragma ignored.
W0521553	[Message]	Nonstandard empty wide character literal treated as L'0'.
W0521561	[Message]	Predefined meaning of "*symbol*" discarded.
W0521564	[Message]	enum qualified name is nonstandard.
W0521565	[Message]	Anonymous union qualifier is nonstandard.
W0521566	[Message]	Anonymous union qualifier is ignored.
W0521570	[Message]	Nonstandard specifier ignored.
W0521607	[Message]	#pragma text must precede the definition of function "*function*".
	[Explanation]	#pragma text must precede the definition of function "*function*". The specification will be ignored.
W0521611	[Message]	#pragma *character string* coflicts with previous #pragma for this function "*function*".
W0521632	[Message]	Unrecognized symbol information "*character string*" is ignored.
W0521635	[Message]	"*variable*" has already been placed into "*section*" section in symbol file. The latter is ignored.
W0521636	[Message]	"*variable*" has already been placed into "*section*" section in symbol file. #pragma is ignored.
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.
W0521651	[Message]	needs multi core device.
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.

W0523060	[Message]	Incompatible section specified
	[Explanation]	The same identifier was declared several times and different sections were specified for individual declarations. Only the first section declaration is valid.
W0523061	[Message]	argument is incompatible with formal parameter of intrinsic function
W0523062	[Message]	return value type does not match the intrinsic function
W0523063	[Message]	"*character string*" has no effect in this version
W0523064	[Message]	Address taken "*variable-name*". It may cause an upset endian indirect reference.
	[Explanation]	The address of an 8-byte variable "*variable name*", which was in the different endian from that specified by the endian option, was acquired. This may lead to an indirect reference with incorrect handling of endian.
W0523068	[Message]	Atomic transfer function is used in "*cpu*"
W0523076	[Message]	Function declarations should have prototype.
	[Explanation]	The function is declared in a different format from the prototype. A function declaration in a format different from the prototype may degrade the efficiency when passing a near pointer as an argument.
W0523077	[Message]	Called function should have prototype.
	[Explanation]	This function call uses a function type without a prototype. When the defining function has a prototype, a mismatch may occur in passing of arguments. The call being made via the function pointer means that there is no prototype for the function pointer type.
W0530809	[Message]	const qualifier for variable "*variable-name*" differs between files.
W0530811	[Message]	Type of symbol "*symbol-name*" differs between files.
W0533003	[Message]	Shift count(*value*) is out of range.
W0533004	[Message]	Result of comparison is always *character string*.
W0533005	[Message]	Division by zero.
W0542101	[Message]	-Xswitch=table option is ignored, because the branch table for switch is overflow.
	[Explanation]	The branch table for switch is overflow. The -Xswitch=table option will be ignored if the difference between the maximum and minimum label values is greater than 8192.
W0544001	[Message]	Alignment of "*section-name*" sections is inconsistent. "*value*" is assumed.
	[Explanation]	There is an error in section naming. The same section name is specified for sections with different alignment numbers.
W0544002	[Message]	Endian of "*section-name*" sections is inconsistent. "*endian type*" is assumed.
	[Explanation]	There is an error in section naming. The same section name is specified for sections in different endian.
W0550001	[Message]	Too many arguments.
	[Action by User]	Check the actual arguments.
W0550002	[Message]	Cannot use *option1* with *option2*, *option2* ignored.
	[Action by User]	Check the option specification.
W0550003	[Message]	"*option*" option needs argument, ignored.
	[Action by User]	Check the option specification parameters.

W0550004	[Message]	Illegal "*option*" option's value, ignored.
	[Action by User]	Check the option specification values.
W0550005	[Message]	Illegal "*option*" option's symbol "*symbol*", ignored.
	[Action by User]	Check the option specification symbols.
W0550006	[Message]	Illegal "*option*" option's argument, ignored.
	[Action by User]	Check the option specification parameters.
W0550007	[Message]	*option*, -C mismatch. ignore -C. output *core* common object.
	[Action by User]	Check the option specification.
W0550008	[Message]	*option* option is not supported for *core* core.
	[Explanation]	*option* option is not supported for *core* core. The option specification will be ignored.
	[Action by User]	Check the option specification.
W0550009	[Message]	Cannot find programmable peripheral I/O registers, ignored -Xprogrammable_io option.
	[Action by User]	Check the option specification.
W0550010	[Message]	Illegal displacement in *inst* instruction.
	[Explanation]	Illegal displacement in inst instruction. Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the displacement value.
W0550011	[Message]	Illegal operand (range error in immediate).
	[Explanation]	Illegal operand (range error in immediate). Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the immediate value.
W0550012	[Message]	Operand overflow.
	[Explanation]	Operand overflow. Only the effective lower-order digits will be recognized as being specified, and the assembly will continue.
	[Action by User]	Check the operand value.
W0550013	[Message]	*register* used as *kind* 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.

W0550016	[Message]	Illegal operand (access width mismatch).
	[Action by User]	Check the operand's internal peripheral I/O 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 4).
	[Explanation]	Illegal operand (immediate must be multiple of 4). The number is rounded down, and assembly continues.
	[Action by User]	Check the operand value.
W0550020	[Message]	Duplicated cpu type, ignored $PROCESSOR.
	[Explanation]	The -C option is given precedence, and the target-device specified by the $PROCESSOR control instruction will be ignored.
	[Action by User]	Check the option specification.
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.
W0550022	[Message]	Duplicated option *option*, ignored.
	[Explanation]	Duplicated option *option*, ignored. The previously specified option will be used. This specification will be ignored.
	[Action by User]	Check the option specification.
W0550023	[Message]	Start address of programmable io is out of range(0x0,*value1*-*value2*),ignored -Xprogrammable_io option.
	[Explanation]	Start address of programmable io is out of range(0x0,*value1*-*value2*),ignored -Xprogrammable_io option. The specified value will be ignored, and the initial value of the device will be used.
	[Action by User]	Check the option values.
W0550024	[Message]	Sorry, -option *option* not implemented, ignored.
	[Action by User]	Check the option specification.
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.
W0550027	[Message]	Illegal *control* value, ignored.
	[Explanation]	The *control* control instruction differs from a previous specification. The previous specification will take precedence, and register modes specified by subsequent *control* control instructions will be ignored.
	[Action by User]	Check the *control* control instruction 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.
W0550029	[Message]	Can not use r0 as destination in mul/mulu in *device-name* core.
	[Action by User]	Check the operand.
W0550030	[Message]	Can not use mul/mulu X,Y,Y format in *device-name* core.
	[Action by User]	Check the operand.
W0550031	[Message]	*identifier* undefined.
	[Action by User]	Check the identifier.
W0550032	[Message]	Cache instruction is used as cll.
	[Action by User]	The use of the cache instruction as cll is not recommended. Use the cll instruction.
W0550605	[Message]	"*path-name*" specified by the "*character string*" option is a file. Specify a folder.
W0550606	[Message]	The folder "*folder-name*" specified by the "*character string*" option is not found.
W0550623	[Message]	The "*character string2*" option is ignored when the "*character string1*" option is specified at the same time.
W0550644	[Message]	"-C" option and "-Xcommon" option is mismatch. Instruction set by "*character string1*" option is ignored. Create common object for "*character string2*" instruction set.
W0550646	[Message]	"*character string1*" specified in the "*character string2* option is not allowed for a preprocessor macro. Recognized only as an assembler symbol
W0550647	[Message]	The "*character string*" option is specifiedmore than once. The latter is valid.
W0550649	[Message]	The "*character string2*" option is ignored when the "*character string1*" option and the "*character string2*" option are inconsistent.
W0550651	[Message]	The "*character string2*" option is ignored when the "*character string1*" option is not specified.
W0550656	[Message]	Device file is not found in the folder specified by the "-Xdev_path" option.
	[Explanation]	Device file is not found in the folder specified by the "-Xdev_path" option. The compiler will search for the device file in the standard device-file folder.
W0551000	[Message]	'.ALIGN' with not 'ALIGN' specified relocatable section.
	[Explanation]	Directive command .ALIGN is written in a section that does not have an ALIGN specification.
	[Action by User]	Check the position where directive command .ALIGN is written. Write an ALIGN specification in the section definition line of a section in which directive command .ALIGN is written.
W0551001	[Message]	Destination address may be changed.
	[Explanation]	The jump address can be a position that differs from an anticipated destination.
	[Action by User]	When writing an address in a branch instruction operand using a location symbol for offset, be sure to write the addressing mode, jump distance, and instruction format specifiers for all mnemonics at locations from that instruction to the jump address.

W0551002	[Message]	Floating point value is out of range.
	[Explanation]	The floating-point value is out of range.
	[Action by User]	Check the floating-point value written in the source code. The value out of range is ignored.
W0551003	[Message]	Location counter exceed.
	[Explanation]	The location counter value has exceeded 0FFFFFFFFh.
	[Action by User]	Check the value of the operand in .ORG. Correct the source code.
W0551004	[Message]	'.ALIGN' size is different.
	[Explanation]	The specified boundary alignment value does not match the other settings.
	[Action by User]	Check the alignment value.
W0551006	[Message]	Data in 'CODE' section align in 4byte.
	[Explanation]	When endian=big is specified, the start address of the data area in the CODE section is aligned to a 4-byte boundary.
W0551007	[Message]	Data size in 'CODE' section align in 4byte.
	[Explanation]	When endian=big is specified, the size of the data area in the CODE section is adjusted to a multiple of 4.
W0551009	[Message]	Multiple symbols.
	[Explanation]	.STACK(stack value setting) is specified multiple times for a single symbol.
W0551010	[Message]	Section attribute mismatch.
	[Explanation]	The specified section attribute does not match the other settings.
W0551011	[Message]	Use PM instruction.
	[Explanation]	A privileged instruction is used.
W0551012	[Message]	Use FPU instruction.
	[Explanation]	A floating-point operation instruction is used.
W0551013	[Message]	Use DSP instruction.
	[Explanation]	A DSP function instruction is used.
W0551014	[Message]	Too many actual macro parameters.
	[Explanation]	There are too many actual macro parameters. Extra macro parameters will be ignored.
W0551015	[Message]	Actual macro parameters are not enough.
	[Explanation]	The number of actual macro parameters is smaller than that of formal macro parameters. The formal macro parameters that do not have corresponding actual macro parameters are ignored.
W0551016	[Message]	'.END' statement is in include file.
	[Explanation]	The include file contains an .END statement. The software will ignore .END as it executes.
	[Action by User]	.END cannot be written in include files. Delete this statement.

W0550032	[Message]	Illegal vector register number, aligned odd register(vrXX) to be even register(vrYY).
	[Explanation]	Odd-numbered vector registers (vr1, vr3, ... vr31) have been specified. The only vector registers that can be specified are even-numbered (vr0, vr2, vr4, ... vr30). Assembly will continue, assuming that even-numbered registers (vr0, r2, vr4, ... vr30) have been specified.
	[Action by User]	Check the vector register specification.
W0560111	[Message]	Multiple inclusion of same file attempted, ignored.
W0560116	[Message]	Input files do not clear what FPU.
W0560205	[Message]	Symbol "*symbol*" has incompatible type in file "*file*".
W0560206	[Message]	Symbol "*symbol*" has different size in file "*file*".
W0560207	[Message]	Symbol "*symbol*" has different align-size in file "*file*".
W0560212	[Message]	Size zero sbss or bss attribute symbol "*symbol*".
W0560214	[Message]	Undefined global symbol "*symbol*" specified with "$EXT_FUNC".
W0560305	[Message]	Section "*section*" with section type(*section type*) not supported, ignored.
W0560308	[Message]	Input files have different register modes. use -Xregmode_info option for more information.
W0560312	[Message]	"*stirng*" option overrides "*section*" section.
W0560313	[Message]	Illegal flash mask option access(file: "*file*" address:*num1* bit:*num2*).
W0560314	[Message]	Section "*section*" alignment must be 4 in internal instruction RAM.
W0560315	[Message]	Section "*section*" attribute is illegal in internal ROM/internal instruction RAM.
W0560316	[Message]	Input files have different register modes.
W0560401	[Message]	Relocation entry in section "*section*" has unknown relocation type(*number*), ignored this entry.
W0560402	[Message]	Can not find numberth symbol table entry for relocation of reference at offset(*offset*) in "*section*" section, this relocation is ignored.
W0560403	[Message]	Relocation entry in relocation section "*section1*" used to relocate section "*section2*" has illegal r_offset(*offset*), ignored.
W0560406	[Message]	Can not find GP-symbol in segment "*segment*" of illegal label reference for local symbol in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*).local symbol is allocated in section "*section1*" (file:*file1*).
W0560408	[Message]	Can not find GP-symbol in segment "*segment*" of illegal label reference for symbol "*symbol*" in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file1*).
W0560410	[Message]	Can not find GP-symbol in section "*section*" of file "*file1*" or illegal label reference for symbol "*symbol*" in file "*file2*" (section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file1*).
W0560412	[Message]	Relocated value(*value*) of relocation entry(file:*file*, section:*section*, offset:*offset*, type:*relocation type*) for branch command become odd value.
W0560413	[Message]	Relocated value(*value*) of relocation entry(file:*file*, section:*section*, offset:*offset*, type:*relocation type*) for load/store command become odd value.
W0560416	[Message]	Symbol "*symbol*" (output section *section1*)is too far from output section "*section2*". (value:*value*, file:*file* input section:*section3*, offset;*offset*, type:*relocation type*)

W0560418	[Message]	Output section "*section1*" is too far from output section "*section2*". (value:*value*, file:*file* input section:*section3*, offset;*offset*, type:*relocation type*)
W0560420	[Message]	Output section "*section1*" overflowed or illegal label reference for symbol "*symbol*" in file "*file*". (value:*value*, input section:*section2*, offset;*offset*, type:*relocation type*). "*symbol*" is allocated in section "*section1*" (file:*file*)
W0560422	[Message]	Output section "*section1*" overflowed or illegal label reference for local symbol in file "*file2*". (value:*value*, input section:*section2*, offset;*offset*, type:*relocation type*). local symbol is allocated in section "*section1*" (file:*file1*)
W0560501	[Message]	No library symbol table, ignored this library file.
W0560625	[Message]	Aligned odd value(*number1*) to be even value(*number2*).
W0560628	[Message]	Segment "*segment*" overflowed highest address of target machine.
W0560639	[Message]	"*stirng*" in segment directive is illegal when "-Xrelinkable_object" option specified, ignored.
W0560640	[Message]	"-Xrelinkable_object" option specified, ignored.
W0560642	[Message]	No LOAD segments exist for mapping input section "*section*" in file "*file*". this section is mapped to non-LOAD *DUMMY* segment with no program header.
W0560643	[Message]	"*stirng*"is illegal in 1pass-mode, ignored, try in 2pass-mode ("-Xtwo_pass_link"option).
W0560644	[Message]	"*stirng*" is illegal when "-Xalign_fill" option specified, ignored value of "*stirng*"
W0560645	[Message]	stringsymbol multiply defined to segment "*segment*", first defined symbol "*symbol*" used.
W0560647	[Message]	stringsymbol multiply defined, first defined symbol "*symbol*" used.
W0560651	[Message]	Segment "*segment*"(*number1-number2*) must be in string-relative-addressable range(*number3-number4*).
W0560653	[Message]	Segment "*segment*"(*number1-number2*) overflowed highest or lowest address of internal memory (*number3-number4*).
W0560654	[Message]	"*stirng*" specified in EP symbol directive, ignored.
W0560655	[Message]	Segment "*segment*" (*number1-number2*) overlaps guarded area (*number3-number4*)
W0560656	[Message]	Segment "*segment*" (*number1-number2*) overlaps programmable peripheral I/O area (*number3-number4*)
W0560659	[Message]	Must be specified in link directive is illegal when address of segment "*segment*" is not specified.
W0560702	[Message]	"*string1*" option is illegal when "*string2*" option is specified, ignored "*string1*" option.
W0560706	[Message]	"*stiring*" option aligned odd value(*value1*) to be even value(*value2*).
W0560709	[Message]	"*stiring1*" option is not supported for "*stiring2*" core.
W0560710	[Message]	Segment sort function is active, because new vector type exist in device file.
W0560711	[Message]	"*stiring*" option is not supported for this device.
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 : "*number*"
	[Explanation]	The level of an abort error message cannot be changed. The specification of *number* is ignored. Only errors at the information/warning/error level can be changed with the change_message option.
W0561005	[Message]	Subcommand file terminated with end option instead of exit option
	[Explanation]	There is no processing specification following the end option. Processing is done with the exit option assumed.
W0561006	[Message]	Options following exit option ignored
	[Explanation]	All options following the exit option is ignored.
W0561007	[Message]	Duplicate option : "*option*"
	[Explanation]	Duplicate specifications of *option* were found. Only the last specification is effective.
W0561008	[Message]	Option "*option*" is effective only in cpu type "*CPU type*"
	[Explanation]	*option* is effective only in *CPU type*. option is ignored.
W0561010	[Message]	Duplicate file specified in option "*option*" : "*file name*"
	[Explanation]	*option* was used to specify the same file twice. The second specification is ignored.
W0561011	[Message]	Duplicate module specified in option "*option*" : "*module*"
	[Explanation]	*option* was used to specify the same module twice. The second specification is ignored.
W0561012	[Message]	Duplicate symbol/section specified in option "*option*" : "*name*"
	[Explanation]	*option* was used to specify the same symbol name or section name twice. The second specification is ignored.
W0561013	[Message]	Duplicate number specified in option "*option*" : "*number*"
	[Explanation]	*option* was used to specify the same error number. Only the last specification is effective.
W0561017	[Message]	Section "*section*" is already defined by -Xrompsec_data option and therefor this section is ignored.
W0561018	[Message]	Section "*section*" is already defined by -Xrompsec_text option and therefor this section is ignored.
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 rename option conflicts between symbol and section
	[Explanation]	*name* specified by the rename option exists as both a section name and as a symbol name. Rename is performed for the symbol name only in this case.

W0561102	[Message]	Symbol "*symbol*" redefined in option "*option*"
	[Explanation]	The symbol specified by *option* has already been defined. Processing is continued without any change.
W0561103	[Message]	Invalid address value specified in option "*option*" : "*address*"
	[Explanation]	*address* specified by *option* is invalid. *address* specification is ignored.
W0561104	[Message]	Invalid section specified in option "*option*" : "*section*"
	[Explanation]	An invalid section is specified in *option*.
	[Action by User]	Confirm the following: (1) The "-output" option does not accept specification of a section that has no initial value. (2) The "-jump_entries_for_pic" option accepts specification of only a code section and no other sections.
W0561110	[Message]	Entry symbol "*sybol*" in entry option conflicts
	[Explanation]	A symbol other than *symbol* specified by the entry option is specified as the entry symbol on compiling or assembling. The option specification is given priority.
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.
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.

W0561180	[Message]	Directive command "*control directive*" is duplicated in "*file*"
	[Explanation]	*control directive* is written in multiple source files. The control directive cannot be written more than once across files.
W0561181	[Message]	Fail to write "*type of output code*"
	[Explanation]	Failed to write *type of output code* to the output file. The output file may not contain the address to which *type of output code* should be output. Type of output code: When failed to write CRC code : "CRC Code"
W0561182	[Message]	Cannot generate vector table section "*section*"
	[Explanation]	The input file contains vector table *section*. The optimizing linker does not create *section* automatically.
W0561183	[Message]	Interrupt number "*vector number*" of "*section*" is defined in input file
	[Explanation]	The vector number specified by the VECTN option is defined in the input file. Processing is continued with priority given on the definition in the input file.
W0561190	[Message]	Section "*section*" was moved other area specified in option "*cpu=<attribute>*"
	[Explanation]	The object size was modified through optimization of access to external variables. Accordingly, *section* in the area specified by the next cpu specification was moved.
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.
W0561192	[Message]	Bss Section "*section name*" is not initialized
	[Explanation]	*section name*, which is a data section without an initial value, cannot be initialized by the initial setup program.
	[Action by User]	Check the address range specified with -cpu and the sizes of pointer variables.
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.
W0561194	[Message]	Section "*section*" in relocation "*file*"-"*section*"-"*offset*" is changed
	[Explanation]	The relocation *section file offset* now refers to a location in the new section created with the division of section.
	[Action by User]	To prevent division, declare the contiguous_section option for *section*.
W0561195	[Message]	Read only data memory domains differ in "*file*"
	[Explanation]	"*file*" with a different memory model is input.
	[Action by User]	Check if the compiler options are correct.
W0561200	[Message]	Backed up file "*file1*" into "*file2*"
	[Explanation]	Input file *file1* was overwritten. A backup copy of the data in the previous version of *file1* was saved in *file2*.
W0561300	[Message]	Option "*option*" is ineffective without debug information
	[Explanation]	There is no debugging information in the input files. The "*option*" has been ignored.
	[Action by User]	Check whether the relevant option was specified at compilation or assembly.

W0561301	[Message]	No inter-module optimization information in input files
	[Explanation]	No inter-module optimization information is present in the input files. The optimize option has been ignored.
	[Action by User]	Check whether the goptimize option was specified at compilation or assembly.
W0561302	[Message]	No stack information in input files
	[Explanation]	No stack information is present in the input files. The stack option is ignored. If all input files are assembler output files, the stack option is ignored.
W0561305	[Message]	Entry address in "*file*" conflicts : "*address*"
	[Explanation]	Multiple files with different entry 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.
W0561321	[Message]	Entry symbol "*symbol*" in "*file*" conflicts
	[Explanation]	Multiple object files containing more than one entry symbol definition were input. Only the entry symbol in the first file input is effective.
W0561322	[Message]	Section alignment mismatch : "*section*"
	[Explanation]	Sections with the same name but different boundary alignments were input. Only the largest boundary alignment specification is effective.
W0561323	[Message]	Section attribute mismatch : "*section*"
	[Explanation]	Sections with the same name but different attributes were input. If they are an absolute section and relative section, the section is treated as an absolute section. If the read/write attributes mismatch, both are allowed.
W0561324	[Message]	Symbol size mismatch : "*symbol*" in "*file*"
	[Explanation]	Common symbols or defined symbols with different sizes were input. A defined symbol is given priority. In the case of two common symbols, the symbol in the first file input is given priority.
W0561325	[Message]	Symbol attribute mismatch : "*symbol*":"*file*"
	[Explanation]	The attribute of *symbol* in *file* does not match the attribute of the same-name symbol in other files.
	[Action by User]	Check the symbol.
W0561326	[Message]	Reserved symbol "*symbol*"is defined in "*file*"
	[Explanation]	Reserved symbol name *symbol* is defined in *file*.
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.

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.
W0561430	[Message]	Cannot generate effective bls file for compiler optimization
	[Explanation]	An invalid bls file was created. This optimization is not available even if optimization of access to external variables (map option) is specified for compilation.
	[Action by User]	The optimization of access to external variables (map option) in the compiler has the following restriction. Check if this restriction is applicable and modify the section allocation. Access to external variables cannot be optimized in some cases if a data section is allocated immediately after a data section when the base option is specified for compilation. Note: The bls file indicates the external symbol allocation information file. It contains the information to be used for the map option of the compiler.
W0561500	[Message]	Cannot check stack size
	[Explanation]	There is no stack section, and so consistency of the stack size specified by the stack option on compiling cannot be checked.
	[Action by User]	To check the consistency of the stack size on compiling, the goptimize option needs to be specified on compiling and assembling.
W0561501	[Message]	Stack size overflow : "*stack size*"
	[Explanation]	The stack section size exceeded the stack size specified by the stack option on compiling.
	[Action by User]	Either change the option used on compiling, or change the program so as to reduce the use of the stack.
W0561502	[Message]	Stack size in "*file*" conflicts with that in another file
	[Explanation]	Different values for stack size are specified for multiple files.
	[Action by User]	Check the options used on compiling.
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.
W0561520	[Message]	"*user option byte/control value for the on-chip debug*" in "*section*" created by device file
W0561521	[Message]	Cannot generate section "*section*"
	[Action by User]	Check the option during compiling and the section specification of a source file.
W0561531	[Message]	"*option*" ignored because Device file with "*tag*" is required
	[Action by User]	Check the device file.
W0562010	[Message]	-Xhex_symtab=global and -Xhex_symtab=all expect -Xhex_format=T.
W0562011	[Message]	-Xhex_symtab=global expect -Xhex_format=T.
W0562012	[Message]	-Xhex_symtab=all expect -Xhex_symtab=global and -Xhex_format=T.

W0562013	[Message]	"*string1*" option overrides "*string2*" option.
W0562015	[Message]	Section "*section*" overflowed lowest address of internal memory.
W0562016	[Message]	Section "*section*" overflowed highest address of internal memory.
W0562017	[Message]	Section "*section*" overflowed lowest address of program memory.
W0562018	[Message]	Section "*section*" is converted from its midst.
W0562019	[Message]	Section "*section*" is converted until its midst.
W0562020	[Message]	The address of convert area exceeds the maximum value of the address that can be expressed in the Intel expanded hex format.
	[Explanation]	This message may be caused by one of the following. (1) Forgot to perform ROMization You may have tried to convert a section allocated to ROM and an internal RAM section into hex simultaneously. Perform ROMization before performing hex conversion (see the "8.5 ROMization" for details about ROMization). (2) The wrong section is being converted to hex Two section addresses being converted to hex simultaneously could be far apart. If you are using multiple ROM, perform hex conversion separately for each one. Also make sure that you are converting the right section to hex, and that the section alignment address is correct. (3) The section itself is too large Hex conversion will fail if the size of the area is too large to be expressed. Either break the section into expressible ranges, or use a different hex format. [Caution] Although the maximum address value is stated as 20 bits, it is actually in the vicinity of 20 bits. The 20-bit Intel extended hex-format address expression that is the cause of this message is calculated from the extended address record in the higher-order address, and the unique data record offset from there. If the higher-order address of the extended address record fits within 20 bits, then the address will be output successfully in Intel extended hex format even if the addition of the subsequent data record offset causes the result to be more than 20 bits, and it will not cause an error.
W0562021	[Message]	The address of convert area exceeds the maximum value of the address that can be expressed in the Motorola S type hex format (standard address).
	[Explanation]	This message may be caused by one of the following. (1) Forgot to perform ROMization You may have tried to convert a section allocated to ROM and an internal RAM section into hex simultaneously. Perform ROMization before performing hex conversion (see the "8.5 ROMization" for details about ROMization). (2) The wrong section is being converted to hex Two section addresses being converted to hex simultaneously could be far apart. If you are using multiple ROM, perform hex conversion separately for each one. Also make sure that you are converting the right section to hex, and that the section alignment address is correct. (3) The section itself is too large Hex conversion will fail if the size of the area is too large to be expressed. Either break the section into expressible ranges, or use a different hex format.

W0562022	[Message]	The start address of convert area exceeds the maximum value of the address that can be expressed in the Intel expanded hex format.
	[Explanation]	This message may be caused by one of the following. (1) The section address is too large The address allocated to the section may be larger than the expressible range. Specify the start address of the area for hex conversion with the -Xhex_offset option, and offset from that address. (2) The value specified via the -Xhex_offset option is invalid It is possible handle the address as an offset from the specified value by specifying the -Xhex_offset option. The offset from this value may be larger than the expressible range. Specify an appropriate value.
W0562023	[Message]	The start address of convert area exceeds the maximum value of the address that can be expressed in the Motorola S type hex format (standard address).
	[Message]	This message may be caused by one of the following. (1) The section address is too large The address allocated to the section may be larger than the expressible range. Specify the start address of the area for hex conversion with the -Xhex_offset option, and offset from that address. (2) The value specified via the -Xhex_offset option is invalid It is possible handle the address as an offset from the specified value by specifying the -Xhex_offset option. The offset from this value may be larger than the expressible range. Specify an appropriate value.
W0562024	[Message]	Too small block length. Length => *length*.
W0562025	[Message]	Too large block length. Length => *length*.
W0562026	[Message]	Block length is set. Length => *length*.
W0562027	[Message]	Symbol block length exceed default value.
W0571600	[Message]	An error occurred during name decoding of "*instance*"
	[Explanation]	"*instance*" was not decoded. The message is output using the encoding name.
W0578306	[Message]	can not open file *file*
W0578307	[Message]	can not close file *file*
W0578308	[Message]	can not read file *file*
W0578309	[Message]	can not write file *file*
W0578310	[Message]	can not seek file *file*
W0578311	[Message]	can not find file *file*
W0578315	[Message]	can not find member *member*
	[Explanation]	Can not find member *member* in the library file.
W0578322	[Message]	this symbol offset not true
	[Explanation]	This symbol offset not true in the library file.
W0592009	[Message]	Failed to open an input file "*file*"
	[Explanation]	If there is a subsequent file specified in the list file, then the applications proceeds to convert that file.

W0592011	[Message]	Input file specified on the command line is ignored when the "-l" option is specified.
	[Explanation]	A list file and input file cannot be specified simultaneously. The application will perform the conversions in the list file specified by the "-l" option, and the input file will be ignored.
W0592012	[Message]	The "-c" option specified on the command line is ignored when it does not match the specification in list file (*file*).
	[Explanation]	The list file and command line have different "-c" option specifications for the input file "*file*" specified in the list file. The conversion specified by the list file will be performed.
W0592017	[Message]	Input file "*file*" is not found.
	[Explanation]	Cannot find input file '*file*'. If there is a subsequent file specified in the list file, then the applications proceeds to convert that file.
W0592103	[Message]	Unrecognized pragma '*string*'.
	[Explanation]	The pre-conversion C-language source file could not be converted, because it contains an illegal #pragma statement.
	[Action by User]	Edit the C-language source file and correct any syntax errors.
W0592104	[Message]	Cannot convert __interrupt/__multi_interrupt inmacro definition.
	[Explanation]	Cannot convert __interrupt and __multi_interrupt described in the macro-definition.
	[Action by User]	The user modify convert __interrupt and __multi_interrupt.
W0592105	[Message]	_rcopy() found in C source.
	[Explanation]	Refer to the _rcopy function from a C source program.
	[Action by User]	Changed to a call from an assembly source, or declare the variable by adding the const qualifier.
W0594100	[Message]	Cannot find file(*file name*). Check that the specified file name is correct
	[Explanation]	The specified file was not found.
	[Action by User]	Specify the file name again.
W0594101	[Message]	Cannot open file to read(*file name*). Check that the file is readable.
	[Explanation]	The specified file is read-protected.
	[Action by User]	Select the file's attribute to make it readable.
W0594102	[Message]	Cannot open file to write(*file name*). Check that both the file and the folder are writable.
	[Explanation]	The specified file is write-protected.
	[Action by User]	Select the file and folder attributes to make it writeable.
W0594103	[Message]	File read error(*file name*).
	[Explanation]	An error occurred while loading the specified file.
	[Action by User]	Make sure that the file is readable.
W0594104	[Message]	File write error(*file name*).
	[Explanation]	An error occurred while writing the specified file.
	[Action by User]	Make sure that the file is writeable.
W0594105	[Message]	Too many characters in one line in file (*file name*:line *number*).

W0594106	[Message]	Too many lines in file(*file name*). The limit is *number*.
W0594107	[Message]	Too long file name(*file name*). The limit is 255.
W0594110	[Message]	Unknown function name (*function name*) in file(*file name*:line *nember*). Only functions explicitly referred to in the project are allowed. Do you want to stop reading?
	[Explanation]	The stack-size specification file specifies a function name not used in the project. Only functions that are defined in or directly called from a C source file can be specified.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594111	[Message]	Too long file name (*file name*) qualifying static function name in file(*file name*:line *number*). The limit is 255. Do you want to stop reading?
	[Explanation]	The name of a file in which a static-function is defined (added in front of the static function in the stack-size specification file) is too long. Use no more than 255 characters.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594112	[Message]	Too long function name (*function name*) in file (*file name*:line *number*). The limit is 1022. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains a function name that is too long. Use no more than 1,022 characters.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.

W0594113	[Message]	Illegal value (*value*) after "ADD=" in file (*file name*:line *number*). Do you want to stop reading?
	[Explanation]	The stack-size specification file contains an invalid added size specification. The added size must be in the format of "ADD=" followed by a decimal number, or a hexadecimal number preceded by "0x" or "0X". The available values are as follows. 0 - 2,147,483,647
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594114	[Message]	Multiple "ADD="s for one function in file (*file name*:line *number*). Only one is allowed. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains an invalid added size specification. An added size can only be specified for one function.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594115	[Message]	Illegal value (*value*) after "RECTIME=" in file (*file name*:line *number*). Do you want to stop reading?
	[Explanation]	The stack-size specification file contains a specification of an invalid number of recursions. The number of recursions must be in the format of "RECTIME=" followed by a positive decimal number, or a positive hexadecimal number preceded by "0x" or "0X".
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.

W0594116	[Message]	Multiple "RECTIME="s for one function in file (*file name*:line *number*). Only one is allowed. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains a specification of an invalid number of recursions. A recursion count can only be specified for one function.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594117	[Message]	Illegal "RECTIME=" for non-recursive function in file (*file name*:line *number*). It shall be specified for recursive functions. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains a specification of an invalid number of recursions. The "RECTIME=" recursion-count specification can only be used with recursive functions.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594118	[Message]	Missing function name after "CALL=" in file (*file name*:line *number*). Do you want to stop reading?
	[Explanation]	The stack-size specification file contains an invalid callee function specification. Specify "CALL=" followed by the function name. Functions that are not used in the project cannot be specified.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.

W0594119	[Message]	Multiple function names after "CALL=" in file (*file name*:line *number*). Place "CALL=" before each functions. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains an invalid callee function specification. "CALL=" can only specify one function. If you wish to specify more than one function, use the "CALL=" specification for each one.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594124	[Message]	Multiple specifications for one function in file (*file name*:line *number1*, line *number2*). Only one is allowed. Do you want to stop reading?
	[Explanation]	The stack-size specification file contains the same function on lines {line-number-1} and {line-number-2}. Delete one of them.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594125	[Message]	Too many characters in one line in file (*file name*:line *number*). The limit is 5119. Do you want to stop reading?
	[Explanation]	The number of characters in line {line-number} of the stack-size specification file exceeds the maximum length (5,119). Reduce the callee function specifications. Even if you register all functions that are called, only the function with the largest stack size will be added to the stack size.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.

W0594126	[Message]	Illegal format in file (*file name*:line *number*). Check the file. Do you want to stop reading?
	[Explanation]	The format of line *line-number* in the stack-size specification file is invalid. Check the file.
	[Action by User]	[Abort]: Cancel processing. [Retry]: Ignore the error and continue loading from the following line. [Ignore]: Ignore the error and continue loading from the following line. Subsequent warnings of this type will be shown in the message display area, but this dialog box will not appear again.
W0594127	[Message]	Can not find stack size specification file for system library functions (*file name*). Default values (the same as *default file name*) are used.
	[Explanation]	No stack-size specification file was found for the standard library. The size information for skcx (same as *default-file-name*) will be used.
	[Action by User]	It may not have been installed correctly; try installing again.
W0594128	[Message]	Illegal format in stack size specification file for system library functions(*file name*:line *number*).
	[Explanation]	The format of the stack-size specification file for the standard library is invalid.
	[Action by User]	It may not have been installed correctly; try installing again.
W0594129	[Message]	Broken stk system file (*file name*). The file was ignored.
	[Explanation]	The stk system file is invalid.
	[Action by User]	A previous change to the stack size may not have been reflected. Check it, and configure it again if it is incorrect.
W0594130	[Message]	Cannot find CX generated assembly language source file. Select [Compile Options] tab in CubeSuite's property panel, and specify [Yes(-Xasm_path)] as [Output assembler source file] property in [Output File] category, then rebuild.
	[Explanation]	The intermediate assembler file was not found.
	[Action by User]	Perform the rebuild after doing the following: register a C source file in CS+; then from the Build Tool Property panel, select the [Compile Options] tab; next, under the [Output File] category, set the [Output assembler source file] property to [Yes(-Xasm_path)].
W0594131	[Message]	Cannot find CX generated assembly language source file (*file name*). Check "Compiler Options" specified for the corresponding C source file and rebuild.
	[Explanation]	The intermediate assembler file was not found.
	[Action by User]	Perform the build after doing the following: from CS+, in the Property panel for the corresponding C source, select the [Individual Compile Options] tab; then under the [Output File] category, set the [Output assembler source file] property to [Yes(-Fs)].

W0594132	[Message]	Illegal format in file(*file name*:line *number*). Check the file
	[Explanation]	The format of line *line-number* in the intermediate assembler file is invalid.
	[Action by User]	The intermediate assembler file generated by CX may be missing. Check the file and rebuild.
W0594133	[Message]	Cannot accept *CC name version* generated assembler source module file. Use *version* or later.
	[Explanation]	The correct stack size cannot be calculated if a call to a runtime library or standard library is included, because the compiler version is out of date.
	[Action by User]	It supports to CX. Rebuild it.
W0594134	[Message]	Assembler source module file for different device (*file name*).
	[Explanation]	File was generated for a different series' compiler. This file cannot be parsed.
W0594135	[Message]	Assembler source module file generated by different version of *CC name*(*file name*).
	[Explanation]	The assembly source module file was compiled using a different compiler version. This file cannot be parsed.
W0594140	[Message]	Illegal project file (*file name*).Check the file.
	[Explanation]	The project file is invalid. Check the file. This error occurs when the project file opened from the menu after startup is invalid.
W0594150	[Message]	Number of lines reached limit (32767) in output.
	[Explanation]	The output result exceeds 32,767 lines.
	[Action by User]	Either change the function selection or output the largest path only.
W0594151	[Message]	Number of characters in one line reached limit(5119) in output.
	[Explanation]	The output result includes a line that is longer than 5,119 characters. Either change the function selection or output in text format. Text-format output will be displayed with line breaks.
W0594160	[Message]	Too many callee functions. The limit is 1024.
	[Explanation]	The number of callee functions specified exceeds the limit. No more than 1,024 functions may be specified. Even if you register all functions that are called, only the function with the largest stack size will be added to the stack size.
W0594161	[Message]	Too many characters in specification for one function. The limit is 5119.
	[Explanation]	The size configuration information exceeds the maximum number of characters (5,119). Reduce the callee function specifications. Even if you register all functions that are called, only the function with the largest stack size will be added to the stack size. The 5,119-character limit includes parameter names and separators (",").

W0594162	[Message]	Illegal value (*value*) for additional margin.
	[Explanation]	An invalid value was specified as the additional size. The added size must be a decimal number, or a hexadecimal number preceded by "0x" or "0X". The available values are as follows. 0 - 2,147,483,647
W0594163	[Message]	Illegal value ({value}) for recursion depth.
	[Explanation]	An invalid value was specified as the number of recursions. Specify a positive number in decimal format, or in hexadecimal format preceded by "0x" or "0X".
W0594164	[Message]	Too big additional margin. The limit is *value*.
	[Explanation]	The value specified as the additional size exceeds the limit (*value*). Change the value. The available values are as follows. 0 - 2,147,483,647
W0594165	[Message]	Total stack size exceeds *value* in function (*function name*). Reduce recursion depth.
	[Explanation]	The amount of stack used by the function exceeds the limit.
	[Action by User]	Reduce the number of recursions.
W0594166	[Message]	Total stack size exceeds *value* in function (*function name*). Reduce stack size in the maximum call chain.
	[Explanation]	The amount of stack used by the function exceeds the limit.
	[Action by User]	Reduce the stack size, starting with the function with the largest path.

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 V850E3v5 G3K Core Specification for Assembler (-Xcpu=g3k Option)

The following instruction sets cannot be specified.

- Hardware multithreading feature instructions
- Virtualization support feature instructions
- MMU control instructions
- SIMD instructions
- Floating-point operation instructions

The above instruction sets cause assemble errors when the V850E3v5 G3K core is specified. If any of them is specified, the assembler outputs the following message.

E0550269 : Illegal mnemonic(cannot use this mnemonic ins RH850 "G3K" core).

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.

```
#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.

```
#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 are as follows:

- Variables with no initial value: .bss section
- Variables with initial value: .data section
- const constants: .const section

To change the allocated section, specify the attribute strings using #pragma section directive.

The relationship between attribute strings and the section generated is as follows.

Attribute Strings	Initial Value	Default Section Name	Section Name Change	Base Register	Access Instruction
r0_disp16	Yes	.zdata	Possible	r0	ld/st 1 instruction
	No	.zbss	Possible	r0	ld/st 1 instruction
r0_disp23	Yes	.zdata23	Possible	r0	ld23/st23 1 instruction
	No	.zbss23	Possible	r0	ld23/st23 1 instruction
r0_disp32	Yes	.data	Possible	r0	movhi+ld/st 2 instruction
	No	.bss	Possible	r0	movhi+ld/st 2 instruction
ep_auto	Yes/No	Automatically selected from among .tdata4, .tdata5, .tdata7, and .tdata8	Impossible	ep	sld/sst 1 instruction
ep_disp4	Yes	.tdata4	Possible	ep	sld/sst 1 instruction
	No	.tbss4	Possible	ep	sld/sst 1 instruction

Attribute Strings	Initial Value	Default Section Name	Section Name Change	Base Register	Access Instruction
ep_disp5	Yes	.tdata5	Possible	ep	sld/sst 1 instruction
	No	.tbss5	Possible	ep	sld/sst 1 instruction
ep_disp7	Yes	.tdata7	Possible	ep	sld/sst 1 instruction
	No	.tbss7	Possible	ep	sld/sst 1 instruction
ep_disp8	Yes	.tdata8	Possible	ep	sld/sst 1 instruction
	No	.tbss8	Possible	ep	sld/sst 1 instruction
ep_disp16	Yes	.edata	Possible	ep	ld/st 1 instruction
	No	.ebss	Possible	ep	ld/st 1 instruction
ep_disp23	Yes	.edata23	Possible	ep	ld23/st23 1 instruction
	No	.ebss23	Possible	ep	ld23/st23 1 instruction
gp_disp16	Yes	.sdata	Possible	gp	ld/st 1 instruction
	No	.sbss	Possible	gp	ld/st 1 instruction
gp_disp23	Yes	.sdata23	Possible	gp	ld23/st23 1 instruction
	No	.sbss23	Possible	gp	ld23/st23 1 instruction
const	Yes	.const	Possible	r0	movhi+ld/st 2 instruction
zconst	Yes	.zconst	Possible	r0	ld/st 1 instruction
zconst23	Yes	.zconst23	Possible	r0	ld23/st23 1 instruction
default	After this statement, any previous #pragma section will be ignored, and the default allocation will be used.				

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
```

When referencing a variable using the #pragma section directive from a function in another file (i.e. reference file), it is necessary to also specify the #pragma section directive in the reference file and to define the affected variable as extern format.

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.1.3 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.1.4 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.1.5 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
static int func(int a, int b) {
        /*Assembler instruction*/
}
```

Remark 1. Note the following when using inline assembly.

- 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.
- RH850 assembly language can use comments starting with the hash character ("#"). Do not use # comments inside functions coded in assembly, because the preprocessor will interpret these as pre-processing directives.
- If you write a label in a function coded in assembly, labels with the same name will be created for each inline expansion. In this situation, use local labels coded in assembly. Although local labels have the same name in the assembly source, the assembler converts them to different names automatically.

Remark 2. See "Describing assembler instruction".

A.2.4 Executing in RAM

A program allocated to external ROM can be copied to internal RAM and executed in internal RAM while linking and after copying if the relative value of each section and each symbol (TP, EP, GP) is not destroyed. Use caution, as some programs can be copied while others cannot.

After resetting, it is copied to internal RAM, and if the program is not changed, then the ROMization function can be used to easily pack the text section.

A.3 Variables (Assembler)

This section explains variables (Assembler).

A.3.1 Defining variables with no initial values

Use the .ds directive in a section with no initial value to allocate area for a variable with no initial value.

```
[label:]        .ds     (size)
```

In order that it may be referenced from other files as well, it is necessary to define the label with the .public directive.

```
.public label name[, absolute-expression]
```

Example Defining variables with no initial values

```
        .dseg   sdata
        .public _val0, 0x4      --Sets _val0 as able to be referenced from other files
        .public _val1, 0x2      --Sets _val1 as able to be referenced from other files
        .public _val2, 0x1      --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[, absolute-expression]
```

Example Defining variable with initial values

```
        .dseg   sbss
        .public _val0, 0x4      --Sets _val0 as able to be referenced from other files
        .public _val1, 0x2      --Sets _val1 as able to be referenced from other files
        .public _val2, 0x1      --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, 0x2         --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	Feb 01, 2015	-	First Edition issued

CC-RH V1.02.00 User's Manual:
Compiler

Publication Date: Rev.1.00 Feb 01, 2015

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2015 Renesas Electronics Corporation. All rights reserved.

Colophon 4.0

CC-RH V1.02.00

Renesas Electronics Corporation

R20UT3280EJ0100