User's Manual

CC-RX V2.00.00

User's Manual: RX Build

Target Device
RX Family

Renesas Electronics
www.renesas.com

Rev.1.00 Jun. 2013

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.

TABLE OF CONTENTS

CHAPTER 1 GENERAL

This chapter introduces the processing of compiling performed by CC-RX, and provides an example of program development using CC-RX.

1.1 Overview

This section describes the components and processing flow of CC-RX.
CC-RX is comprised of the 6 executable files listed below.

(1) ccrx: Compile driver

(2) rxc: Compiler

(3) asrx: Assembler

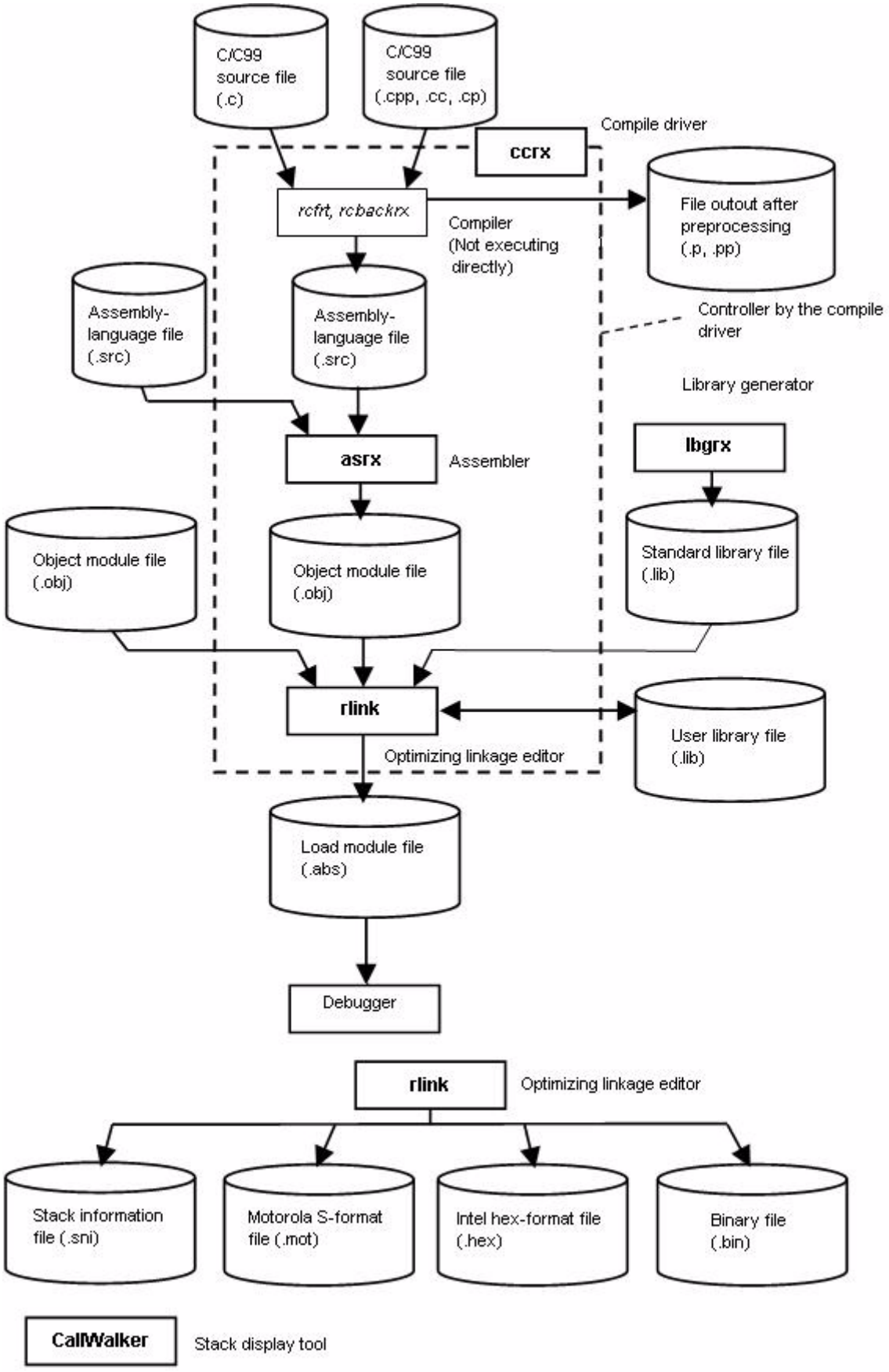
(4) rlink: Optimizing linkage editor

(5) lbgrx: Library generator

(6) CallWalker: Stack display tool

CC-RX Flow illustrates the CC-RX processing flow.

Figure 1-1. CC-RX Processing Flow



1.2 Copyrights

This LLVM-based software was developed in compliance with the LLVM Release License. Copyrights of other software components are owned by Renesas Electronics Corporation.

CHAPTER 2 BUILD OUTPUT LISTS

This chapter describes the format and other aspects of files output by a build via each command.

2.1 Assemble List File

This section covers the contents and format of the assemble list file output by the assembler.

The source list file contains the compilation and assembly results. Table 3.1 shows the structure and contents of the source list.

Table 2-1. Structure and Contents of Source List

No	Output Information	Contents	Suboption[Note]	When -show Option is not Specified
1	Source information	C/C++ source code corresponding to assembly source code	-show=source	Not output
2	Object information	Machine code used in object programs and the assembly source code	None	Output
3	Statistics information	Total number of errors, number of source program lines, and size of each section	None	Output
4	Command specification information	File names and options specified by the command	None	Output

Note Valid when the -listfile option is specified.

2.1.1 Source Information

The source information is included in the object information when the -show**=source** option is specified. For an example of source information, refer to the next section, Object Information.

2.1.2 Object Information

Figure 3.1 shows an example of object information output.

```
* RX FAMILY ASSEMBLER V2.00.00  [15 Feb 2013] *  SOURCE LIST  Mon Feb 18 20:15:19 2013
(1)      (2)               (3)   (4)
LOC.     OBJ.              0XMDA SOURCE STATEMENT

                                 ;RX Family C/C++ Compiler (V2.00.00  [15 Feb 2013])  18-Feb-
2013 20:15:19

                                 ;*** CPU TYPE ***

                                 ;-CPU=RX600

                                 ;*** COMMAND PARAMETER ***

                                 ;-output=src=sample.src
                                 ;-listfile
                                 ;-show=source
                                 ;sample.c

                                 .glb_x
                                 .glb_y
                                 .glb_func02
                                 .glb_func03
                                 .glb_func01
                                 (5)      (6)
                                 ;LineNo. C-SOURCE STATEMENT

                                 .SECTIONP,CODE
00000000                         _func02:
                                 .STACK_func02=12
                                 ;       1 #include "include.h"
                                 ;       2 int func01(int);
                                 ;       3 int func03(int);
                                 ;       4
                                 ;       5 int func02(int z)
00000000 6E67                    PUSHM R6-R7
00000002 EF16                    MOV.L R1, R6
                                 ;       6 {
                                 ;       7     x = func01(z);
00000004 05rrrrrr              A BSR _func01
00000008 FB72rrrrrrrr            MOV.L #_x, R7
0000000E E371                    MOV.L R1, [R7]
                                 ;       8     if (z == 2) {
00000010 6126                    CMP #02H, R6
00000012 18                    S BNE L12
00000013                         L11:; bb3
                                 ;       9         x++;
00000013 6211                    ADD #01H, R1
00000015 08                    S BRA L13
00000016                         L12:; bb6
                                 ;      10     } else {
                                 ;      11         x = func03(x + 2);
00000016 6221                    ADD #02H, R1
00000018 39rrrr                W BSR _func03
0000001B                         L13:; bb13
0000001B E371                    MOV.L R1, [R7]
                                 ;      12     }
                                 ;      13     return x;
                                 ;      14 }
0000001D 3F6702                  RTSD #08H, R6-R7
00000020                         _func03:
                                 .STACK_func03=4
                                 ;      15
                                 ;      16 int func03(int p)
                                 ;      17 {
                                 ;      18     return p+1;
00000020 6211                    ADD #01H, R1
                                 ;      19 }
00000022 02                      RTS
                                 .SECTIOND,ROMDATA,ALIGN=4
00000000                         _y:
00000000 01000000                .lword00000001H
                                 .END
```

Item Number	Description
(1)	Location information (LOC.) Location address of the object code that can be determined at assembly.

<table>
<tr><th>Item Number</th><th>Description</th></tr>
<tr><td>(2)</td><td>Object code information (OBJ.)
Object code corresponding to the mnemonic of the source code.</td></tr>
<tr><td>(3)</td><td>Line information (0XMDA)
Results of source code processing by the assembler. The following shows the meaning of each symbol.
<table>
<tr><th>0</th><th>X</th><th>M</th><th>D</th><th>A</th><th>Description</th></tr>
<tr><td>0-30</td><td></td><td></td><td></td><td></td><td>Shows the nesting level of include files.</td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td>Shows the line where the condition is false in conditional assembly when -show=conditions is specified.</td></tr>
<tr><td></td><td></td><td>M</td><td></td><td></td><td>Shows the line expanded from a macro instruction when -show=expansions is specified.</td></tr>
<tr><td></td><td></td><td>D</td><td></td><td></td><td>Shows the line that defines a macro instruction when -show=definitions is specified.</td></tr>
<tr><td></td><td></td><td></td><td>S</td><td></td><td>Shows that branch distance specifier S is selected.</td></tr>
<tr><td></td><td></td><td></td><td>B</td><td></td><td>Shows that branch distance specifier B is selected.</td></tr>
<tr><td></td><td></td><td></td><td>W</td><td></td><td>Shows that branch distance specifier W is selected.</td></tr>
<tr><td></td><td></td><td></td><td>A</td><td></td><td>Shows that branch distance specifier A is selected.</td></tr>
<tr><td></td><td></td><td></td><td></td><td>*</td><td>Shows that a substitute instruction is selected for a conditional branch instruction.</td></tr>
</table></td></tr>
<tr><td>(4)</td><td>Source information (SOURCE STATEMENT)
Contents of the assembly-language source file.</td></tr>
<tr><td>(5)</td><td>C/C++ source line number (LineNo.)</td></tr>
<tr><td>(6)</td><td>C/C++ source statement (C-SOURCE STATEMENT)
C/C++ source statement output when the -show=source option is specified.</td></tr>
</table>

2.1.3 Statistics Information

The following figure shows an example of statistics information output.

```
Information List (1)
TOTAL ERROR(S)    00000
TOTAL WARNING(S)  00000
TOTAL LINE(S)     00071  LINES
Section List (2)
Attr         Size               Name
CODE     0000000047(0000002FH)  P
ROMDATA  0000000004(00000004H)  D
```

Item Number	Description
(1)	Numbers of error messages and warning messages, and total number of source lines
(2)	Section information (section attribute, size, and section name)

2.1.4 Compiler Command Specification Information

The file names and options specified on the command line when the compiler is invoked are output. The compiler command specification information is output at the beginning of the list file. The following figure shows an example of command specification information output.

```
;*** CPU TYPE *** (1)
;-CPU=RX600
;*** COMMAND PARAMETER *** (2)
;-output=src=C:\tmp\elp1894\sample.src
;-nologo
;-show=source
;sample.c
```

Item Number	Description
(1)	Selected microcomputer
(2)	File names and options specified for the compiler

2.1.5 Assembler Command Specification Information

The file names and options specified on the command line when the assembler is invoked are output. The assembler command specification information is output at the end of the list file. The following figure shows an example of command specification information output.

```
Cpu Type  (1)
-CPU=RX600
Command Parameter  (2)
-output=sample.obj
-nologo
-listfile=sample.lst
```

Item Number	Description
(1)	Microcomputer selected for the assembler
(2)	File names and options specified for the assembler

2.2 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.

2.2.1 Structure of Linkage List

Table 3-3 shows the structure and contents of the linkage list.

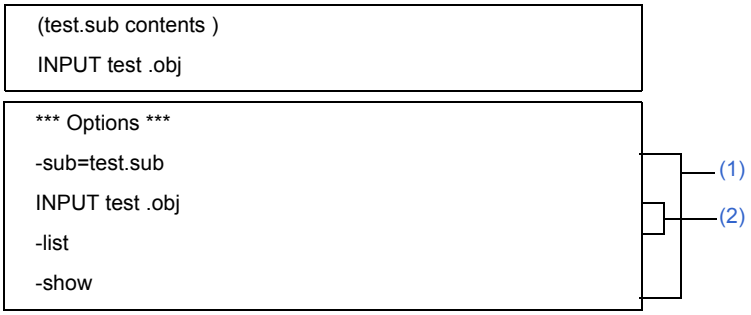
Table 2-2. Structure and Contents of Linkage List

No	Output Information	Contents	When -show Option[Note] is Specified	When -show Option is not Specified
1	Option information	Option strings specified by a command line or subcommand	None	Output
2	Error information	Error messages	None	Output
3	Linkage map information	Section name, start/end addresses, size, and type	None	Output
4	Symbol information	Static definition symbol name, address, size, and type in the order of address	-show =symbol	Not output
		When -show=reference is specified: Symbol reference count and optimization information in addition to the above information	-show =reference	Not output
5	Symbol deletion optimization information	Symbols deleted by optimization	-show =symbol	Not output
6	Cross-reference information	Symbol reference information	-show =xreference	Not output
7	Total section size	Total sizes of RAM, ROM, and program sections	-show=total_size	Not output
8	Vector information	Vector numbers and address information	-show=vector	Not output
9	CRC information	CRC calculation result and output addresses	None	Always output when the CRC option is specified

Note The -show option is valid when the **list** option is specified.

2.2.2 Option Information

The option strings specified by a command line or a subcommand file are output. The following figure shows an example of option information output when **rlink** -subcommand**=test.sub** -list -show is specified.



Item Number	Description
(1)	Outputs option strings specified by a command line or a subcommand in the specified order.
(2)	Subcommand in the **test.sub** subcommand file.

2.2.3 Error Information

Error messages are output. The following figure shows an example of error information output.

```
*** Error Information ***
** E0562310 (E) Undefined external symbol "strcmp" referred to in "test.obj"  (1)
```

Item Number	Description
(1)	Outputs an error message.

2.2.4 Linkage Map Information

The start and end addresses, size, and type of each section are output in the order of address. The following figure shows an example of linkage map information output.

```
*** Mapping List ***
(1)                              (2)       (3)       (4)   (5)
SECTION                          START     END       SIZE  ALIGN
P
                                 00001000  00001000     1  1
C
                                 00001004  00001007     4  4
D_2
                                 00001008  000014dd   4d6  2
B_2
                                 000014de  000050b3   3bd6 2
```

Item Number	Description
(1)	Section name
(2)	Start address

Item Number	Description
(3)	End address
(4)	Section size
(5)	Section boundary alignment value

2.2.5 Symbol Information

When -show**=symbol** is specified, the addresses, sizes, and types of externally defined symbols or static internally defined symbols are output in the order of address. When -show**=reference** is specified, the symbol reference counts and optimization information are also output. The following figure shows an example of symbol information output.

```
*** Symbol List ***
SECTION=(1)
                (3)         (4)          (5)
FILE=(2)        START       END          SIZE
  (6)           (7)         (8)          (9)                (10)    (11)
  SYMBOL        ADDR        SIZE         INFO               COUNTS  OPT
SECTION=P
FILE=test.obj
                00000000    00000428         428
  _main
                00000000           2     func ,g               0
  _malloc
                00000000          32     func ,l               0
FILE=mvn3
                00000428    00000490          68
  $MVN#3
                00000428           0     none ,g               0
```

Item Number	Description
(1)	Section name
(2)	File name
(3)	Start address of a section included in the file indicated by (2) above
(4)	End address of a section included in the file indicated by (2) above
(5)	Section size of a section included in the file indicated by (2) above
(6)	Symbol name
(7)	Symbol address
(8)	Symbol size

Item Number	Description
(9)	Symbol type as shown below 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)	Symbol reference count only when -show=reference is specified. * is output when show=reference is not specified.
(11)	Optimization information as shown below. ch: Symbol modified by optimization cr: Symbol created by optimization mv: Symbol moved by optimization

2.2.6 Symbol Deletion Optimization Information

The size and type of symbols deleted by symbol deletion optimization (-optimize**=symbol_delete**) are output.
The following figure shows an example of symbol deletion optimization information output.

```
*** Delete Symbols ***
(1)                             (2)    (3)
SYMBOL                          SIZE   INFO
  _Version
                                   4   data ,g
```

Item Number	Description
(1)	Deleted symbol name
(2)	Deleted symbol size
(3)	Deleted symbol type as shown below Data type func: Function name data: Variable name Declaration type g: External definition l: Internal definition

2.2.7 Cross-Reference Information

The symbol reference information (cross-reference information) is output when -show**=xreference** is specified. The following figure shows an example of cross-reference information output.

```
*** Cross Reference List ***
(1)  (2)        (3)             (4)         (5)
No   Unit Name  Global.Symbol   Location    External Information
0001 a
     SECTION=P  _func
                                00000100
                _func1
                                00000116
                _main
                                0000012c
                _g
                                00000136
     SECTION=B
                _a
                                00000190    0001(00000140:P)
                                            0002(00000178:P)
                                            0003(0000018c:P)
0002 b
     SECTION=P
                _func01
                                00000154    0001(00000148:P)
                _func02
                                00000166    0001(00000150:P)
0003 c
     SECTION=P
                _func03
                                00000184
```

Item Number	Description
(1)	Unit number, which is an identification number in object units
(2)	Object name, which specifies the input order at linkage
(3)	Symbol name output in ascending order of allocation addresses for every section
(4)	Symbol allocation address, which is a relative value from the beginning of the section when -form=relocate is specified
(5)	Address of an external symbol that has been referenced Output format: <Unit number> (<address or offset in section>:<section name>)

2.2.8 Total Section Size

The total sizes of ROM, RAM, and program sections are output. The following figure shows an example of total section size output.

```
*** Total Section Size ***
RAMDATA SECTION :       00000660 Byte(s)  (1)
ROMDATA SECTION :       00000174 Byte(s)  (2)
PROGRAM SECTION :       000016d6 Byte(s)  (3)
```

Item Number	Description
(1)	Total size of RAM data sections
(2)	Total size of ROM data sections
(3)	Total size of program sections

2.2.9 Vector Information

The contents of the variable vector table are output when -show**=vector** is specified. The following figure shows an example of vector information output.

```
*** Variable Vector Table List ***
(1)     (2)
NO.     SYMBOL/ADDRESS
 0      $fdummy
 1      $fa
 2      00ff8800
 3      $fdummy
   :
<Omitted>
```

Item Number	Description
(1)	Vector number
(2)	Symbol. When no symbol is defined for the vector number, the address is output.

2.2.10 CRC Information

The CRC calculation result and output address are output when the CRC option is specified.

```
*** CRC Code ***
CODE    : cb0b      (1)
ADDRESS : 00007ffe  (2)
```

Item Number	Description
(1)	CRC calculation result
(2)	Address where the CRC calculation result is output

2.3 Library List

This section covers the contents and format of the library list output by the optimizing linkage editor.

2.3.1 Structure of Library List

Table 3.4 shows the structure and contents of the library list.

Table 2-3. Structure and Contents of Library List

No	Output Information	Contents	Suboption[Note]	When -show Option is not Specified
1	Option information	Option strings specified by a command line or subcommand	-	Output
2	Error information	Error messages	-	Output
3	Library information	Library information	-	Output
4	Information of modules, sections, and symbols within library	Module within the library	-	Output
		When **show=symbol** is specified: List of symbol names in a module within the library	-show=symbol	Not output
		When **show=section** is specified: Lists of section names and symbol names in a module within the library	-show=section	Not output

Note All options are valid when the -list option is specified.

2.3.2 Option Information

The option strings specified by a command line or a subcommand file are output.

The following figure shows an example of option information output when **rlink** -subcommand **= test.sub** -list -show is specified.

```
(test.sub contents)
form     library
in       adhry.obj
output   test.lib
```

```
*** Options ***
-sub=test.sub
form    library
in      adhry.obj
output  test.lib
-list
-show
```

(1)
(2)

Item Number	Description
(1)	Outputs option strings specified by a command line or a subcommand in the specified order.

Item Number	Description
(2)	Subcommand in the test.sub subcommand file.

2.3.3 Error Information

Messages for errors or warnings are output.

The following figure shows an example of error information output.

```
*** Error Information ***
** W0561200 (W) Backed up file "main.lib" into "main.lbk"  (1)
```

Item Number	Description
(1)	Outputs a warning message.

2.3.4 Library Information

The library type is output.

The following figure shows an example of library information output.

```
*** Library Information ***
LIBRARY NAME =test.lib  (1)
CPU=RX620               (2)
ENDIAN=Big              (3)
ATTRIBUTE=system        (4)
NUMBER OF MODULE =1     (5)
```

Item Number	Description
(1)	Library name
(2)	CPU name
(3)	Endian type
(4)	Library file attribute: either system library or user library
(5)	Number of modules within the library

2.3.5 Module, Section, and Symbol Information within Library

A list of modules within the library is output.

When -show**=symbol** is specified, the symbol names in a module within the library are listed. When -show**=section** is specified, the section names and symbol names in a module within the library are listed.

The following figure shows an output example of module, section, and symbol information within a library.

```
*** Library List ***
(1)             (2)
MODULE          LAST UPDATE
  (3)
  SECTION
   (4)
   SYMBOL
adhry
                29-Feb-2000 12:34:56
  P
   _main
   _Proc0
   _Proc1
  C
  D
   _Version
  B
   _IntGlob
   _CharGlob
```

Item Number	Description
(1)	Module name
(2)	Module definition date If the module is updated, the latest module update date is displayed.
(3)	Section name within a module
(4)	Symbol within a section

2.4 S-Type and HEX File Formats

This section describes the S-type files and HEX files that are output by the optimizing linkage editor.

2.4.1 S-Type File Format

Figure 2-1. S-Type File Format

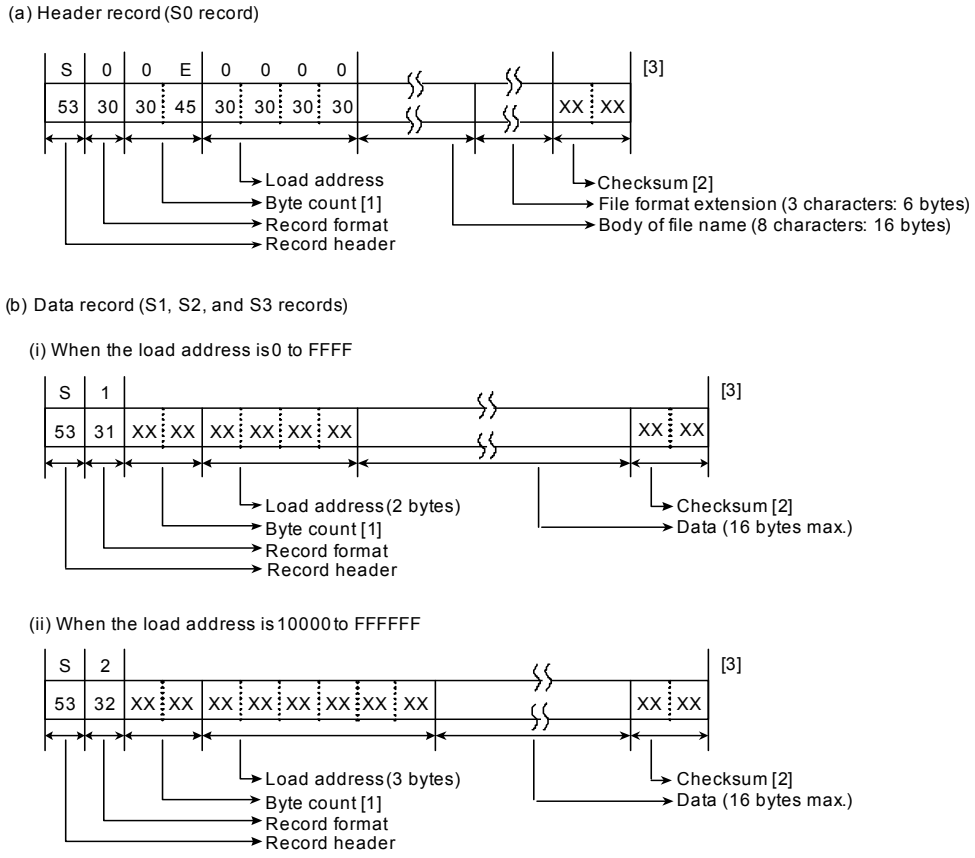
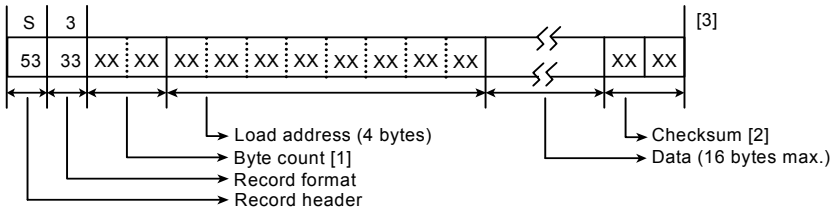


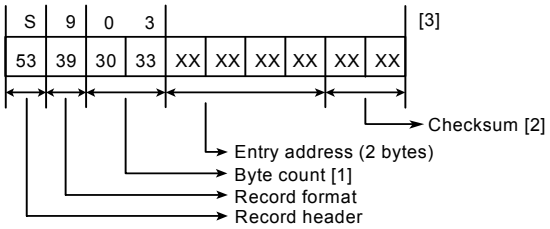
Figure 2-2. S-Type File Format (cont)

(iii) When the load address is 1000000 to FFFFFFFF

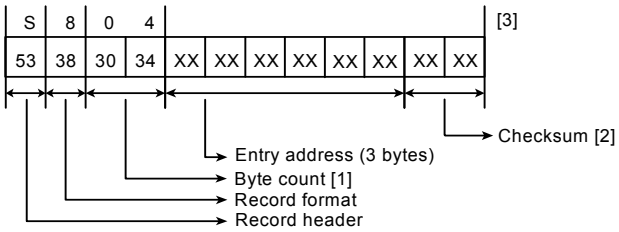


(c) End record (S9, S8, and S7 records)

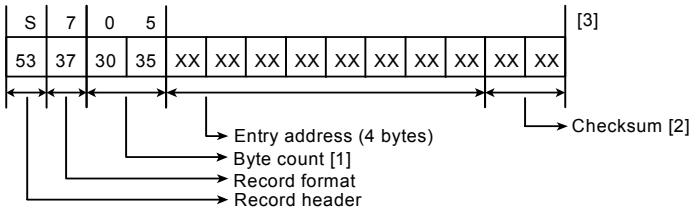
(i) When the entry address is 0 to FFFF



(ii) When the entry address is 10000 to FFFFFF



(iii) When the entry address is 1000000 to FFFFFFFF



Notes: [1] The number of bytes from the load address (or the entry address) to the checksum.

[2] 1's complement of the sum of the byte count and the data between the checksum and the byte count, in byte units.

[3] A new-line character is added immediately after the checksum.

2.4.2 HEX File Format

The execution address of each data record is obtained as described below.

(1) Segment address

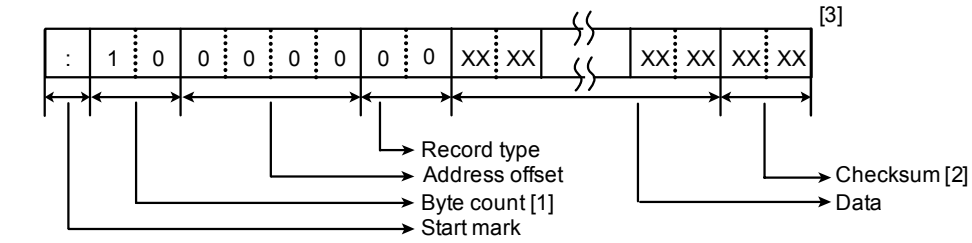
(Segment base address << 4) + (Address offset of the data record)

(2) Linear address

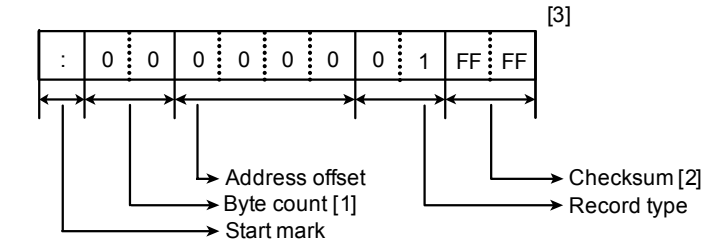
(Linear base address << 16) + (Address offset of the data record)

Figure 2-3. HEX File Format

(a) Data record (00 record)



(b) End record (01 record)



(c) Expansion segment address record(02 record)

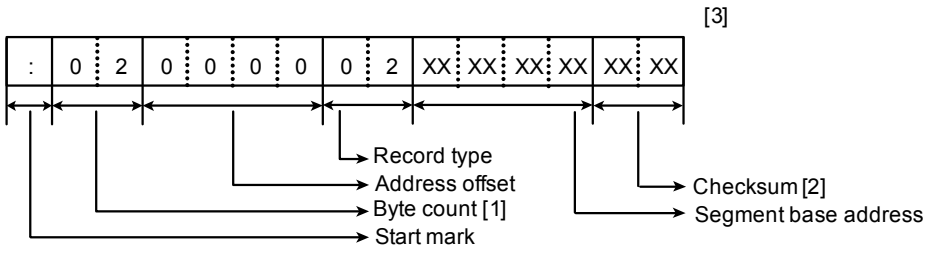
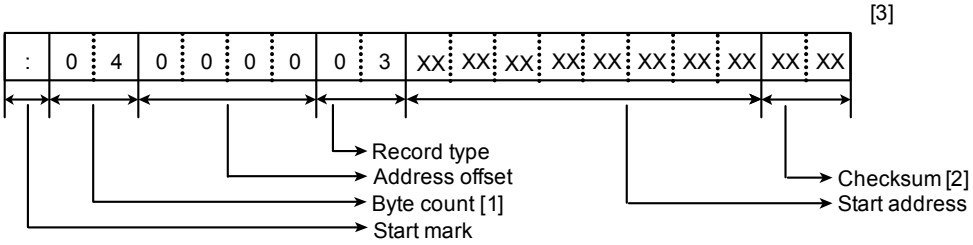
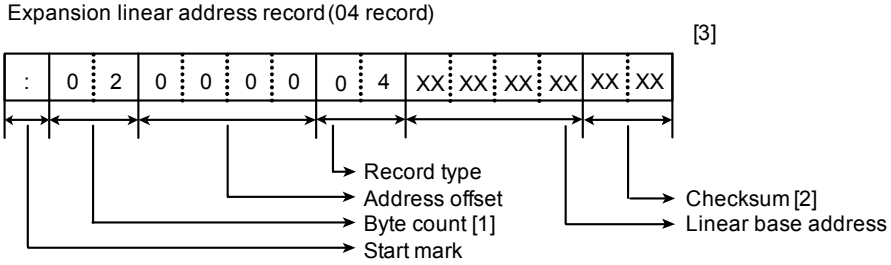


Figure 2-4. HEX File Format (cont)

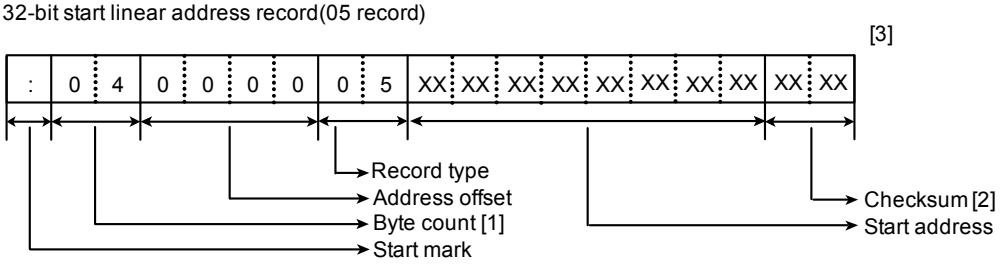
(d) Start address record (03 record)



(e) Expansion linear address record (04 record)



(f) 32-bit start linear address record(05 record)



Notes: [1] The number of bytes from the byte following the record type to the previous byte of the checksum.

[2] 2's complement of the sum of the byte count and the data between the byte count and checksum, in hexadecimal (lower 8 bits are valid).

[3] A new-line character is added immediately after the checksum.

APPENDIX A COMMAND REFERENCE

This appendix describes the detailed specifications of each command included in the build tool.

A.1 RX Family C/C++ Compiler

The RX family C/C++ compiler generates a file executable in the target system from the source program written in C language, C99 language, C++ language, or assembly language.

In this compiler, a single driver controls multiple phases from preprocessing to linkage.

The following describes processing in each phase.

(1) Compiler

This processes preprocessing directives, comments, and optimization for the C source program and generates an assembly-language source file.

(2) Preprocessor

This processes the preprocessing directives in the source program.

Only when the -P option is specified, it outputs the preprocessed file.

(3) Parsing section

This parses the C source program and then converts it to the internal data representation for the compiler.

(4) Optimizing section

This optimizes the internal data representation converted from the C source program.

(5) Code generating section

This converts the internal data representation to an assembly-language source program.

(6) Assembler

This converts the assembly-language source program to machine-language instructions and generates a relocatable object module file.

(7) Optimizing linkage editor

This links object module files, link directive files, and library files and generates an object file (load module file) executable in the target system.

A.1.1 Input/Output Files

The following shows the files input to and output from the RX family C/C++ compiler.

Table A-1. Input/Output Files for the RX Family C/C++ Compiler

File Type	Extension	I/O	Description
C source program file	.c	Input	A source file written in C99 language. This file is created by the user.
C++ source program file	.cpp, .cp, and .cc	Input	A source file written in C++ language. This file is created by the user.
Include file	Optional	Input	A file referenced by the source file and written in C, C99, C++, or assembly language. This file is created by the user.
Preprocessor expansion file for the C program	.p	Output	A file output as a result of preprocessing applied to an input C-language or the C99-language source program. An ASCII image file. This is output when the -output=prep option is specified.
Preprocessor expansion file for the C++ program	.pp	Output	A file output as a result of preprocessing applied to an input C++-language source program. An ASCII image file. This is output when the -output=prep option is specified.
Assembly-source program file	.src	Output	An assembly-language file generated from a C, C99, or C++ source file through compilation.
	.src	Input	A source file written in assembly language.
List file for the assembly program	.lst	Output	A list file containing the assembly result information. This is output when the -listfile option is specified. The output contents are selected with the -show option.
Relocatable object program file	.obj	Output	An ELF-format file that contains the machine-language information, the relocation information about the allocation addresses of machine-language instructions, and symbol information.
Absolute load module file	.abs	Output	An ELF-format file for the object code generated as a result of linkage. This is an input file when a hex file is output.
Linkage list file	.map	Output	A list file containing the linkage result information. This is output when the -list option is specified. The output contents are selected with the -show option.
Library file	.lib	Output	A file where multiple object module files are registered.
Library list file	.lbp	Output	A list file containing the result information of generation of the library. This is output when the -list option is specified. The output contents are selected with the -show option.
Library backup file	.lbk	Output	File type for saving the contents of original library files before they are overwritten by the library generator.
Hex file (Motorola S-format file)	.mot	Output	A Motorola S-format file in the hex format converted from the load module file.
Hex file (Intel (expansion) hex format file)	.hex	Output	An Intel (expansion) file in the hex format converted from the load module file.
Hex file (binary format file)	.bin	Output	A binary file in the hex format converted from the load module file.

File Type	Extension	I/O	Description
Stack information file	.sni	Output	A stack information file. This is output when the -stack option is specified.
Debugging information file	.dbg	Output	A debugging information file. This is output when the -sdebug option is specified.
Object file including a definition specified with a file having extension td	.rti	Output	An object file including a definition specified with a file having extension td.
Calling information file	.cal	Output	A calling information file. This is output by CallWalker.
External symbol assignment information file	.bls	Output	An external symbol assignment information file. This is output at linkage when the -map option is specified.
	.bls	Input	An external symbol assignment information file. This is specified as an input file for the -map option at compilation.
Jump table file (assembly language)	.jmp	Output	An assembler source file for the jump table that branches the external definition symbol. This is output when the -jump_entries_for_pic option is specified.
Symbol address file (assembly language)	.fsy	Output	An assembler source file that describes the external definition symbol in an assembler directive. This is output when the -fsymbol option is specified.
C++ language function support file	.td, .ti, .pi, and .ii	Output	An information file that supports the C++ language function.

A.1.2 Operating Instructions

This section describes how to operate the RX family C/C++ compiler.

(1) Operating Tools

(a) Compiler (ccrx)

ccrx is the startup command for the compile driver.
Compilation, assemble, and linkage can be performed using this command.
When the extension of an input file is ".s", ".src", ".S", or ".SRC", the compiler interprets the file as an assembly-language file (.src, .s) and initiates the assembler.
A file with an extension other than those above is compiled as a C/C++ source file (.c, .cpp).
[Command description format]

```
ccrx [Δ<option> …][Δ<file name>[ Δ<option> …] …]
       <option>: -<option>[=<suboption>[=<suboption>]][, …]
```

(b) Assembler (asrx)

asrx is the startup command for the assembler.
[Command description format]

```
asrx [Δ<option> …][ Δ<file name>[ Δ<option> …] …]
       <option>: -<option>[=<suboption>][, …]
```

(c) Optimizing Linkage Editor (rlink)

rlink is the startup command for the optimizing linkage editor.

The optimizing linkage editor has the following functions as well as the linkage processing.

- Optimizes relocatable files at linkage
- Generates and edits library files
- Converts files into Motorola S type files, Intel hex type files, and binary files

[Command description format]

```
rlink [Δ<option> …][ Δ<file name>[ Δ<option> …] …]
         <option>: -<option>[=<suboption>][, …]
```

(d) Library Generator (lbgrx)

lbgrx is the startup command for the library generator.

[Command description format]

```
lbgrx [Δ<option> …]
        <option>: -<option>[=<suboption>][, …]
```

(2) Command Description Examples

(a) Compilation, Assemble, and Linkage by One Command

Perform all steps below by a single command.

- Compile C/C++ source files (tp1.c and tp2.c) in **ccrx**.
- After compilation, assemble the files in **asrx**.
- After assemble, link the files in **rlink** to generate an absolute file (tp.abs).

[Command description]

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.c tp2.c
```

Remarks 1. When the output type specification of the **output** option is changed to **-output=sty**, the file after linkage will be generated as a Motorola S type file.

2. An intermediate file generated during the absolute file generation process (assembly-language file or relocatable file) is not saved. Only a file of the type specified by the **output** option is to be generated.

3. In order to specify assemble options and linkage options that are valid for only the assembler and optimizing linkage editor in **ccrx**, use the **-asmcmd**, **-lnkcmd**, **-asmopt**, and **-lnkopt** options.

4. Object files that are to be linked are allocated from address 0. The order of the sections is not guaranteed. In order to specify the allocation address or section allocation order, specify options for the optimizing linkage editor using the **-lnkcmd** and **-lnkopt** options.

(b) Compilation and Assemble by One Command

Perform all steps below by a single command, and initiate the linker with another command to generate tp.abs.

- Compile C/C++ source files (tp1.c and tp2.c) in **ccrx**.
- After compilation, assemble the files in **asrx** to generate relocatable files (tp1.obj and tp2.obj).

[Command description]

```
ccrx -cpu=rx600 -output=obj tp1.c tp2.c
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

Remarks 1. When the **-output=obj** option is specified in **ccrx**, **ccrx** generates relocatable files.

2. In order to change relocatable file names, their C/C++ source files have to be input in **ccrx**, one file each.

3. When the **form** option in **rlink** is changed to **-form=sty**, the file after linkage will be generated as a Motorola S type file.

(c) Compilation, Assemble, and Linkage by Separate Commands

Individually perform each step below by a single command.

- Compile C/C++ source files (tp1.c and tp2.c) in **ccrx** to generate assembly-language files (tp1.src and tp2.src).
- Assemble the assembly-language files (tp1.src and tp2.src) in **asrx** to generate relocatable files (tp1.obj and tp2.obj).
- Link the relocatable files (tp1.obj and tp2.obj) in **rlink** to generate an absolute file (tp.abs).

[Command description]

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

Remark When the **-output=src** option is specified in **ccrx**, **ccrx** generates assembly-language files.

(d) Assemble and Linkage by One Command

Perform all steps below by a single command.

- Assemble assembly-language files (tp1.src and tp2.src) in **asrx**.
- After assemble, link the files in **rlink** to generate an absolute file (tp.abs).

[Command description]

```
ccrx -cpu=rx600 -output=abs=tp.abs tp1.src tp2.src
```

Remark Object files that are to be linked are allocated from address 0. The order of the sections is not guaranteed. In order to specify the allocation address or section allocation order, specify options for the optimizing linkage editor using the **-lnkcmd** and **-lnkopt** options.

(e) Assemble and Linkage by Separate Commands

Individually perform each step below by a single command.

- Assemble assembly-language files (tp1.src and tp2.src) in **asrx** to generate relocatable files (tp1.obj and tp2.obj).
- Link the relocatable files (tp1.obj and tp2.obj) in **rlink** to generate an absolute file (tp.abs).

[Command description 1]

```
ccrx -cpu=rx600 -output=obj tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

[Command description 2]

```
asrx -cpu=rx600 tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

(3) Environment Variables (Command Prompt)

Environment variables are listed below.

Table A-2. Environment Variables

No.	Environment Variable	Description	Default When Specification is Omitted
1	path	Specifies a storage directory for the execution file.	Specification cannot be omitted.
2	BIN_RX	Specifies the directory in which ccrx is stored.	<ccrx storage directory> Specification cannot be omitted when the lbgrx command is used.
3	CPU_RX	Specifies the CPU type. <CPU type> RX600 RX200	No value is set when specification is omitted.
4	INC_RX	Specifies a directory in which an include file of the compiler is stored.	<ccrx storage directory> \..\include
5	INC_RXA	Specifies a directory in which an include file of the assembler is stored.	No value is set when specification is omitted.
6	TMP_RX	Specifies a directory in which a temporary file is generated.	%TEMP% when the ccrx command is used.
7	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	Specifies a default library name for the optimizing linkage editor. Libraries which are specified by a library option are linked first. Then, if there is an unresolved symbol, the default libraries are searched in the order of 1, 2, 3.	No value is set when specification is omitted.
8	HLINK_TMP	Specifies a folder in which the optimizing linkage editor generates temporary files. If HLNK_TMP is not specified, the temporary files are created in the current folder.	No value is set when specification is omitted.
9	HLINK_DIR	Specifies an input file storage folder for the optimizing linkage editor. The search order for files which are specified by the input or library option is the current folder, then the folder specified by HLNK_DIR. However, when a wild card is used in the file specification, only the current folder is searched.	No value is set when specification is omitted.

A.1.3 Options

This section describes the options for the RX family C/C++ compiler in each processing phase.

Compile phase: Refer to (1) Compile Options.

Assembly phase: Refer to (2) Assembler Command Options.

Link phase: Refer to (3) Optimizing Linkage Editor (rlink) Options.

Library generation phase: Refer to (4) Library Generator Options.

(1) Compile Options

The types and explanations for options of the compile phase are shown below.

Classification	Option	Description
Source Options	-lang	Specifies the language to assume in compiling the source file.
	-include	Specifies the names of folders that hold include files.
	-preinclude	Specifies the names of files to be included at the head of each compiling unit.
	-define	Specifies macro definitions.
	-undefine	Specifies disabling of predefined macros.
	-message	Information-level messages are output.
	-nomessage	Specifies the numbers of information-level messages to be disabled.
	-change_message	Changes the levels of compiler output messages.
	-file_inline_path	Specifies the names of folders that hold files for inter-file inline expansion.
	-comment	Selects permission for comment (/* */) nesting.
	-check	Checks compatibility with an existing program.
	-misra2004	Checks the source code against the MISRA-C: 2004 rules.
	-ignore_files_misra	Selects files that will not be checked against the MISRA-C: 2004 rules.
	-check_language_extension	Enables complete checking against the MISRA-C: 2004 rules for parts of the code where this would otherwise be suppressed due to use of an extended specification.
Object Options	-output	Selects the output file type.
	-noline	Selects the non-output of **#line** in preprocessor expansion.
	-debug	Debugging information is output to the object files.
	-nodebug	Debugging information is not output to the object files.
	-section	Changes section names to be changed.
	-stuff	Variables are allocated to sections that match their alignment values.
	-nostuff	Alignment values of variables are ignored in allocating the variables to sections.
	-instalign4	Instructions at branch destinations are aligned with 4-byte boundaries.
	-instalign8	Instructions at branch destinations are aligned with 8-byte boundaries.
	-noinstalign	Instructions at branch destinations have no specific alignment.
	-nouse_div_inst	Generates code in which no DIV, DIVU, or FDIV instructions are used for division and modular division.

Classification	Option	Description
List Options	-listfile	A source list file is output.
	-nolistfile	A source list file is not output.
	-show	Specifies the contents of the source list file.
Optimize Options	-optimize	Selects the optimization level.
	-goptimize	Outputs additional information for inter-module optimization.
	-speed	Optimization is with emphasis on execution performance.
	-size	Optimization is with emphasis on code size.
	-loop	Specifies a maximum number for loop-expansion.
	-inline	Inline expansion is processed automatically.
	-noinline	Inline expansion is not processed automatically.
	-file_inline	Specifies a file for inter-file inline expansion.
	-case	Selects the method of expansion for **switch** statements.
	-volatile	External variables are handled as if they are all **volatile** qualified.
	-novolatile	External variables are handled as if none of them have been declared **volatile**.
	-const_copy	Enables constant propagation of **const** qualified external variables.
	-noconst_copy	Disables constant propagation of **const** qualified external variables.
	-const_div	Divisions and remainders of integer constants are converted into instruction sequences.
	-noconst_div	Divisions and remainders of integer constants are not converted into instruction sequences.
	-library	Selects the method for the execution of library functions.
	-scope	Selects division of the ranges for optimization into multiple sections before compilation.
	-noscope	Selects non-division of the ranges for optimization into multiple sections before compilation.
	-schedule	Pipeline processing is considered in scheduling instructions.
	-noschedule	Scheduling is not applied to instruction execution.
	-map	All access to external variables is optimized.
	-smap	Access to external variables is optimized as defined in the file to be compiled.
	-nomap	Access to external variables is not optimized.
	-approxdiv	Division of floating-point constants is converted into multiplication.
	-enable_register	Variables with the **register** storage class specification are given preference for allocation to registers.
	-simple_float_conv	Part of the type conversion processing between the floating-point type and the integer type is omitted.

Classification	Option	Description
Optimize Options	-fpu	Floating-point calculation instructions are used.
	-nofpu	Floating-point calculation instructions are not used.
	-alias	Optimization is performed in consideration of the types of data indicated by pointers.
	-float_order	The orders of operations in floating-point expressions are modified for optimization.
	-ip_optimize	Selects global optimization.
	-merge_files	The results of compiling multiple source files are output to a single object file.
	-whole_program	Makes the compiler perform optimization on the assumption that all source files have been input.

Classification	Option	Description
Microcontroller Options	-cpu	Selects the microcontroller type.
	-endian	Selects the endian type.
	-round	Selects the rounding method for floating-point constant operations.
	-denormalize	Selects the operation when denormalized numbers are used to describe floating-point constants.
	-dbl_size	Selects the precision of the **double** and **long double** types.
	-int_to_short	Replaces the **int** type with the **short** type and the **unsigned int** type with the **unsigned short** type.
	-signed_char	Variables of the **char** type are handled as **signed char**.
	-unsigned_char	Variables of the **char** type are handled as **unsigned char**.
	-signed_bitfield	The sign bits of bit-fields are taken as **signed**.
	-unsigned_bitfield	The sign bits of bit-fields are taken as **unsigned**.
	-auto_enum	Selects whether or not the sizes for enumerated types are automatically selected.
	-bit_order	Selects the order of bit-field members.
	-pack	Specifies one as the boundary alignment value for structure members and class members.
	-unpack	Aligns structure members and class members to the alignment boundaries for the given data types.
	-exception	Enables the exception handling function.
	-noexception	Disables the exception handling function.
	-rtti	Selects enabling or disabling of C++ runtime type information (**dynamic_cast** or **typeid**).
	-fint_register	Selects a general register for exclusive use with the fast interrupt function.
	-branch	Selects the maximum size or no maximum size for branches.
	-base	Specifies the base registers for ROM and RAM.
	-patch	Selects avoidance or non-avoidance of a problem specific to the CPU type.
	-pic	Enables the PIC function.
	-pid	Enables the PID function.
	-nouse_pid_register	The PID register is not used in code generation.
	-save_acc	The contents of ACC are saved and restored in interrupt functions.
Assemble and Linkage Options	-asmcmd	Specifies a subcommand file for **asrx** options.
	-lnkcmd	Specifies a subcommand file for **rlink** options.
	-asmopt	Specifies **asrx** options.
	-lnkopt	Specifies **rlink** options.

Classification	Option	Description
Other Options	-logo	Selects the output of copyright information.
	-nologo	Selects the non-output of copyright information.
	-euc	The character codes of input programs are interpreted as EUC codes.
	-sjis	The character codes of input programs are interpreted as SJIS codes.
	-latin1	The character codes of input programs are interpreted as ISO-Latin1 codes.
	-utf8	The character codes of input programs are interpreted as UTF-8 codes.
	-big5	The character codes of input programs are interpreted as BIG5 codes.
	-gb2312	The character codes of input programs are interpreted as GB2312 codes.
	-outcode	Selects the character coding for an output assembly-language file.
	-subcommand	Specifies a file for including command options.

Source Options

The following source options are available.

-lang

[Format]

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

- [Default]
  If this option is not specified, the compiler will compile the program file as a C++ source file when the extension is **cpp**, **cc**, or **cp**, and as a C (C89) source file for any other extensions. However, if the extension is **src** or **s**, the program file is handled as an assembly-language file regardless of whether this option is specified.

[Description]

- This option specifies the language of the source file.
- When the **lang=c** option is specified, the compiler will compile the program file as a C (C89) source file.
- When the **lang=cpp** option is specified, the compiler will compile the program file as a C++ source file.
- When the **lang=ecpp** option is specified, the compiler will compile the program file as an Embedded C++ source file.
- When the **lang=c99** option is specified, the compiler will compile the program file as a C (C99) source file.

[Remarks]

- The Embedded C++ language specification does not support a **catch**, **const_cast**, **dynamic_cast**, **explicit**, **mutable**, **namespace**, **reinterpret_cast**, **static_cast**, **template**, **throw**, **try**, **typeid**, **typename**, **using**, multiple inheritance, or virtual base class. If one of these classes is written in the source file, the compiler will display an error message. Always specify the **lang=ecpp** option when using an EC++ library.

-include

[Format]

```
-include=<path name>[,...]
```

[Description]

- This option specifies the name of the path to the folder that stores the include file.
- Multiple path names can be specified by separating them with a comma (,).
- The system include file is searched for in the order of the folders specified by the **include** option, the folders specified by environment variable **INC_RX**, and the folders specified by environment variable **BIN_RX**.
- The user include file is searched for in the order of the folders containing source files to be compiled, the folders specified by the **include** option, the folders specified by environment variable **INC_RX**, and the folders specified by environment variable **BIN_RX**.

[Remarks]

- If this option is specified for more than one time, all specified path names are valid.

-preinclude

[Format]

```
-preinclude=<file name>[,...]
```

[Description]

- This option includes the specified file contents at the head of the compiling unit. Multiple file names can be specified by separating them with a comma (,).
- If there is more than one folder specified by the **preinclude** option, search is performed in turn starting from the leftmost folder.

[Remarks]

- If this option is specified for more than one time, all specified files will be included.

-define

[Format]

```
-define=<sub>[,...]
        <sub>: <macro name> [= <string>]
```

[Description]

- This option provides the same function as **#define** specified in the source file.
- <string> can be defined as a macro name by specifying <macro name>=<string>.
- When only <macro name> is specified as a suboption, the macro name is assumed to be defined. Names or integer constants can be written in <string>.

[Remarks]

- If the macro name specified by this option has already been defined in the source file by **#define**, **#define** takes priority.
- If this option is specified for more than one time, all specified macro names are valid.

-undefine

[Format]

```
-undefine=<sub>[,...]
          <sub>: <macro name>
```

[Description]

- This option invalidates the predefined macro of <macro name>.
- Multiple macro names can be specified by separating them with a comma (,).

[Remarks]

- For the specifiable predefined macros, refer to Predefined Macros.
- If this option is specified for more than one time, all specified macro names will be undefined.

-message

[Format]

```
-message
```

[Description]

- This option outputs the information-level messages.

[Remarks]

- Message output from the assembler or optimizing linkage editor cannot be controlled by this option. Message output from the optimizing linkage editor can be controlled by using the **lnkcmd** option to specify the **message** or **nomessage** option of the optimizing linkage editor.

-nomessage

[Format]

```
-nomessage [= <error number> [- <error number>][,...]
```

[Description]

- When the **nomessage** option is specified, output of the information-level messages is disabled. When an error number is specified as a suboption, the output of the specified information-level message will be disabled. Multiple error numbers can be specified by separating them with a comma (,).
- A range of error numbers to be disabled can be specified by using a hyphen (-), that is, in the form of <error number>-<error number>.
- Error numbers are specified by the five lower-order digits (i.e. five digits from the right) of message numbers without the prefix "M" (information).
  Example: To change the level of information message M0523009
  -nomessage=23009

[Remarks]

- Message output from the assembler or optimizing linkage editor cannot be controlled by this option. Message output from the optimizing linkage editor can be controlled by using the **lnkcmd** option to specify the **message** or **nomessage** option of the optimizing linkage editor.
- If the **nomessage** option is specified for more than one time, output for all specified error numbers will be disabled.
- This option is only specifiable for messages with number 0510000 to 0549999 (including the component number).
- This option can only be used to suppress the output of messages 0520000 to 0529999. The output of other messages is not suppressed even if their numbers are specified with this option. If you wish to suppress the output of such messages, also use **-change_message** to change them to information messages.

-change_message

[Format]

```
-change_message = <sub>[,...]
                  <sub>: <error level>[=<error number>[- <error number>][,...]]
                         <error level>: { information | warning | error }
```

[Description]

- This option changes the message level of information-level and warning-level messages.
- Multiple error numbers can be specified by separating them with a comma (,).
- Error numbers are specified by the five lower-order digits (i.e. five digits from the right) of the message numbers without the prefix "M" (information) or "W" (warning).
  Example: To change the level of information message M0523009
  -change_message=error=23009
- Although this option may change the types of some messages (e.g. error (E) or warning (W)), the meaning of the message indicated by the component or message number remains the same.

[Example]

```
change_message=information=error number
```

- Warning-level messages with the specified error numbers are changed to information-level messages.

```
change_message=warning=error number
```

- Information-level messages with the specified error numbers are changed to warning-level messages.

```
change_message=error=error number
```

- Information-level and warning-level messages with the specified error numbers are changed to error-level messages.

```
change_message=information
```

- All warning-level messages are changed to information-level messages.

```
change_message=warning
```

- All information-level messages are changed to warning-level messages.

```
change_message=error
```

- All information-level and warning-level messages are changed to error-level messages.

[Remarks]

- The output of messages which have been changed to information-level messages can be disabled by the **nomessage** option.

- Message output from the assembler or optimizing linkage editor cannot be controlled by this option. Message output from the optimizing linkage editor can be controlled by using the **lnkcmd** option to specify the **message** or **nomessage** option of the optimizing linkage editor.
- If this option is specified for more than one time, all specified error numbers are valid.
- Only the levels of warning and information messages can be controlled by this option. Specification of the option for a message not at these levels is ignored.
- This option is not usable to control the level of MISRA2004 detection messages (labeled M) that appear when the **misra2004** option has been specified.

-file_inline_path

[Format]

```
-file_inline_path=<path name>[,...]
```

[Description]

- This option is not available in V.2.00. Any specification of this option will simply be ignored and will not lead to an error due to compatibility with former versions.

-comment

[Format]

```
-comment = { nest | nonest }
```

[Description]

- When **comment=nest** is specified, nested comments are allowed to be written in the source file.
- When **comment=nonest** is specified, writing nested comments will generate an error.
- The default for this option is **comment=nonest**.

[Example]

- When **comment=nest** is specified, the compiler handles the above line as a nested comment; however, when **comment=nonest** is specified, the compiler assumes (1) as the end of the comment.

```
/* This is an example of /* nested */ comment */
                                    (1)
```

-check

[Format]

```
-check = { nc | ch38 | shc }
```

[Description]

- This option checks the specified options and source file parts which will affect the compatibility when this compiler uses a C/C++ source file that has been coded for the R8C and M16C family C compilers, H8, H8S, and H8SX family C/C++ compilers, and SuperH family C/C++ compilers.
- For **check=nc**, the compatibility with the R8C and M16C family C compilers is checked. Checking will be for the following options and types:
- Options: signed_char, signed_bitfield, bit_order=left, endian=big, and dbl_size=4
- inline, enum type, #pragma BITADDRESS, #pragma ROM, #pragma PARAMETER, and asm()
- Assignment of a constant outside the **signed short** range to the **int** or **signed int** type or assignment of a constant outside the **unsigned short** range to the **int** or **unsigned int** type while **-int_to_short** is not specified
- Assignment of a constant outside both of the **signed short** and **unsigned short** ranges to the **long** or **long long** type
- Comparison expression between a constant outside the **signed short** range and the **int**, **short**, or **char** type (except the **signed char** type)
- For **check=ch38**, the compatibility with the H8, H8S, and H8SX family C/C++ compilers is checked. Checking will be for the following options and types:
- Options: unsigned_char, unsigned_bitfield, bit_order=right, endian=little, and dbl_size=4
- __asm and #pragma unpack
- Comparison expression with a constant greater than the maximum value of **signed long**
- Assignment of a constant outside the **signed short** range to the **int** or **signed int** type or assignment of a constant outside the **unsigned short** range to the **int** or **unsigned int** type while **-int_to_short** is not specified
- Assignment of a constant outside both of the **signed short** and **unsigned short** ranges to the **long** or **long long** type
- Comparison expression between a constant outside the signed short range and the **int**, **short**, or **char** type (except the **signed char** type)
- For **check=shc**, the compatibility with the SuperH family C/C++ compilers is checked. Checking will be for the following options and types:
- Options: unsigned_char, unsigned_bitfield, bit_order=right, endian=little, dbl_size=4, and round=nearest
- #pragma unpack
- **volatile** qualified variables
- Confirm the following notes for the displayed items.
- Options: The settings which are not defined in the language specification and depend on implementation differ in each compiler. Confirm the settings of the options that were output in a message.
- Extended specifications: There is a possibility that extended specifications will affect program operation. Confirm the descriptions on the extended specifications that were output in a message.

[Remarks]

- When **dbl_size=4** is enabled, the results of type conversion related to floating-point numbers and the results of library calculation may differ from those in the R8C and M16C family C compilers, H8, H8S, and H8SX family C/C++ compilers, and SuperH family C/C++ compilers. When **dbl_size=4** is specified, this compiler handles **double** type and **long double** type as 32 bits, but the R8C and M16C family C compilers (**fdouble_32**), H8, H8S, and H8SX family C/C++ compilers (**double=float**), and SuperH family C/C++ compilers (**double=float**) handle only **double** type as 32 bits.
- The result of a binary operation (addition, subtraction, multiplication, division, comparison, etc.) with **unsigned int** type and **long** type operands may differ from that in the SuperH family C/C++ compilers. In this compiler, the types of the operands are converted to the **unsigned long** type before operation. However, in the SuperH family C/C++ compilers (only when **strict_ansi** is not specified), the types of the operands are converted to the **signed long long** type before operation.
- The data size of reading from and writing to a **volatile** qualified variable may differ from that in the SuperH family C/C++ compilers. This is because a **volatile** qualified bit field may be accessed in a size smaller than that of the declaration type in this compiler. However, in the SuperH family C/C++ compilers, a **volatile** qualified bit field is accessed in the same size as that of the declaration type.
- This option does not output a message regarding allocation of structure members and bit field members. When an allocation-conscious declaration is made, refer to 3.1.4 Internal Data Representation and Areas in User's Manual: RX Coding.
- In the R8C and M16C family C compilers (**fextend_to_int** is not specified), the generated code has been evaluated without performing generalized integer promotion by a conditional expression. Accordingly, operation of such a code may differ from a code generated by this compiler.

-misra2004

[Format]

```
-misra2004 = {
   all
   | apply=<rule number>[,<rule number>,...]
   | ignore=<rule number>[,<rule number>,...]
   | required
   | required_add=<rule number>[,<rule number>,...]
   | required_remove=<rule number>[,<rule number>,...]
   | <filename> }
```

[Description]

- This option enables checking against the MISRA-C: 2004 rules and to select specific rules to be used.
- When **misra2004=all**, the compiler checks the source code against all of the rules that are supported.
- When **misra2004=apply=<rule number>[,<rule number>,...]**, the compiler checks the source code against the rules with the selected numbers.
- When **misra2004=ignore=<rule number>[,<rule number>,...]**, the compiler checks the source code against the rules other than those with the selected numbers.
- When **misra2004=required**, the compiler checks the source code against the rules of the "required" type.
- When **misra2004=required_add=<rule number>[,<rule number>,...]**, the compiler checks the source code against the rules of the "required" type and the rules with the selected numbers.
- When **misra2004=required_remove=<rule number>[,<rule number>,...]**, the compiler checks the source code against the rules other than those with the selected numbers among the rules of the "required" type.
- When **misra2004=<filename>**, the compiler checks the source code against the rules with the numbers written in the specified file. One rule number is written per line in the file. Each rule number must be specified by using a decimal value and a period (".").
- When checking of a line of code against the MISRA-C: 2004 rules leads to detection of a violation, a message in the following format will appear.
  Filename (line number): M0523028 Rule number: Message
- When **-misra2004=<filename>** is used more than once, only the last specification is valid.

[Remarks]

- -**misra2004** can be used multiple times. If two or more types (which follow **-misra2004=**) are specified, however, only the type of the last specification is valid.
  ... -misra2004=ignore=2.2 -misra2004=apply=2.3
  -misra2004=required_add=4.1 -misra2004=apply=4.2
  -misra2004=apply=5.2 ...
  In this example, **ignore**, **apply**, and **required_add** are specified, but only **apply** (used in the last two cases) is valid. The compiler will check the source code against rules 4.2 and 5.2.
- When the number of an unsupported rule is specified for **<rule number>**, the compiler detects error C0523031 and stops the processing.
- When the file specified in **misra2004=<filename>** cannot be opened, the compiler detects error C0523029. When rule numbers are not extractable from the specified file, the compiler detects error C0523030. Processing by the compiler stops in both cases.

- This option is ignored when cpp, c99, or ecpp is selected for the **lang** option or when **output=prep** is specified at the same time.

- This option supports the MISRA-C: 2004 rules listed below.
  [Required]
  2.2 2.3
  4.1 4.2
  5.2 5.3 5.4
  6.1 6.2 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 11.5
  12.3 12.4 12.5 12.7 12.8 12.9 12.10 12.12
  13.1 13.3 13.4
  14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9 14.10
  15.1 15.2 15.3 15.4 15.5
  16.1 16.3 16.5 16.6 16.9
  18.1 18.4
  19.3 19.6 19.8 19.11 19.14 19.15
  20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12
  [Not required]
  5.5 5.6
  6.3
  11.3 11.4
  12.1 12.6 12.11 12.13
  13.2
  17.5
  19.7 19.13
- For source programs that use extended functions such as **#pragma**, checking against these rules will be suppressed under some conditions. For details, refer to the section on the **check_language_extension** option.

-ignore_files_misra

[Format]

```
-ignore_files_misra=<filename>[,<filename>,...]
```

[Description]

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

[Remarks]

- If a single option is specified more than once in the command line, all specifications are valid.
- This option is ignored when the -misra2004 option has not been specified.
- <filename> is ignored when the specified file is not to be compiled.

-check_language_extension

[Format]

```
-check_language_extension
```

[Description]

- This option enables complete checking against the MISRA-C: 2004 rules for parts of the code where it would otherwise be suppressed due to individual extensions from the C/C++ language specification.
- With the default misra2004 option, the compiler does not proceed with checking against the MISRA-C: 2004 rules under the condition given below. To enable complete checking, specify the **check_language_extension** option.
- Condition:
  - The function has no prototype declaration (rule 8.1) and **#pragma entry** or **#pragma interrupt** is specified for it.

[Example]

```
#pragma interrupt vfunc
extern void service(void);
void vfunc(void)
{
    service();
}
```

- Function **vfunc**, for which **#pragma interrupt** is specified, has no prototype declaration. Even when this function is compiled with **-misra2004=all** specified, the message on rule 8.1 is not displayed unless the **check_language_extension** option is specified.

[Remarks]

This option is ignored when the **-misra2004** option has not been specified.

Object Options

The following object options are available.

- -output
- -noline
- -debug
- -nodebug
- -section
- -stuff
- -nostuff
- -instalign4
- -instalign8
- -noinstalign
- -nouse_div_inst

-output

[Format]

```
-output = <sub> [=<file name>]
          <sub>: { prep | src | obj | abs | hex | sty }
```

- [Default]
  The default for this option is **output=obj**.

[Description]

- This option specifies the output file type.
- The suboptions and output files are shown in the following table.
- If no **<file name>** is specified, a file will be generated with an extension, that is shown in the following table, appended to the source file name input at the beginning.

Table A-3. Suboption Output Format

Suboption	Output File Type	Extension When File Name is Not Specified
prep	Source file after preprocessed	C (C89, C99) source file: p C++ source file: pp
src	Assembly-language file	src
obj	Relocatable file	obj
abs	Absolute file	abs
hex	Intel hex type file	hex
sty	Motorola S type file	mot

Note Relocatable files are files output from the assembler.
Absolute files, Intel hex type files, and Motorola S type files are files output from the optimizing linkage editor.

[Remarks]

- An intermediate file used to generate a file of the specified type is stored in the specified folder; however, when no folder has been specified, the intermediate file is stored in the current folder.

-noline

[Format]

```
-noline
```

[Description]

- This option disables **#line** output during preprocessor expansion.

[Remarks]

- This option is validated when the **output=prep** option has not been specified.

-debug

[Format]

```
-debug
```

[Description]

- When the **debug** option is specified, debugging information necessary for C-source debugging is output. The **debug** option is valid even when an optimize option is specified.

-nodebug

[Format]

```
-nodebug
```

[Description]

- When the **nodebug** option is specified, no debugging information is output.

-section

[Format]

```
-section = <sub>[,...]
           <sub>: { P = <section name> |
                    C = <section name> |
                    D = <section name> |
                    B = <section name> |
                    L = <section name> |
                    W = <section name> }
```

[Description]

- This option specifies the section name.
- **section=P=<section name>** specifies the section name of a program area.
- **section=C=<section name>** specifies the section name of a constant area.
- **section=D=<section name>** specifies the section name of an initialized data area.
- **section=B=<section name>** specifies the section name of an uninitialized data area.
- **section=L=<section name>** specifies the section name of a literal area.
- **section=W=<section name>** specifies the section name of a **switch** statement branch table area.
- <section name> must be alphabetic, numeric, underscore (_), or $. The first character must not be numeric.

[Remarks]

- The default for this option is **section=P=P,C=C,D=D,B=B,L=L,W=W**.
- In the same way as in V. 1.00, if you want to output the literal area in the **C** section rather than output a separate **L** section, select **section=L=C**.
- Except for changing the **L** section to the same section name as that of the **C** section, the same section name cannot be specified for the sections for different areas.
- For the translation limit of the section name length, refer to Translation Limits.

-stuff

[Format]

```
-stuff
```

[Description]

- When the **stuff** option is specified, all variables are allocated to 4-byte, 2-byte, or 1-byte boundary alignment sections depending on the alignment value (see table B-4).

Table A-4. Correspondences between Variables and Their Output Sections When stuff Option is Specified

Variable Type	Alignment Value for Variable	Section to Which Variable Belongs
const qualified variables	4	C
	2	C_2
	1	C_1
Initialized variables	4	D
	2	D_2
	1	D_1
Uninitialized variables	4	B
	2	B_2
	1	B_1
switch statement branch table	4	W
	2	W_2
	1	W_1

- **C**, **D**, and **B** are the section names specified by the **section** option or **#pragma section**. **W** is the section name specified by the **section** option. The data contents allocated to each section are output in the order they were defined, except that variables that do not have the initial value are output after those that have the initial value in section C.

[Example]

```
int a;
char b=0;
const short c=0;
struct {
        char x;
        char y;
} ST;
```

```
        .SECTION                C_2,ROMDATA,ALIGN=2
        .glb        _c
_c:
        .word                   0000H
        .SECTION                D_1,ROMDATA
        .glb        _b
_b:
        .byte                   00H
        .SECTION                B,DATA,ALIGN=4
        .glb        _a
_a:
        .blkl        1
        .SECTION                B_1,DATA
        .glb        _ST
_ST
        .blkb        2
```

[Remarks]

- The **stuff** option has no effect for sections other than **B**, **D**, **C**, and **W**.

-nostuff

[Format]

```
-nostuff [= <section type>[,...]]
           <section type>: { B | D | C | W }
```

[Description]

- When the **nostuff** option is specified, the compiler allocates the variables belonging to the specified <section type> to 4-byte boundary alignment sections. When **<section type>** is omitted, variables of all section types are applicable.
- **C**, **D**, and **B** are the section names specified by the **section** option or **#pragma** section. **W** is the section name specified by the section option. The data contents allocated to each section are output in the order they were defined, except that variables that do not have the initial value are output after those that have the initial value in section C.

[Example]

```
int a;
char b=0;
const short c=0;
struct {
        char x;
        char y;
} ST;
```

```
        .SECTION                C,ROMDATA,ALIGN=4
        .glb        _c
_c:
        .word                   0000H
        .SECTION                D,ROMDATA,ALIGN=4
        .glb        _b
_b:
        .byte                   00H
        .SECTION                B,DATA,ALIGN=4
        .glb        _a
_a:
        .blkl        1

        .glb        _ST
_ST
        .blkb        2
```

[Remarks]

- The **nostuff** option cannot be specified for sections other than **B**, **D**, **C**, and **W**.

-instalign4

[Format]

```
-instalign4[={loop|inmostloop}]
```

[Description]

- This option aligns instructions at branch destinations.
- When the **instalign4** option is specified, the instruction at the location address is aligned to the 4-byte boundary.
- Instruction alignment is performed only when the instruction at the specified location exceeds the address which is a multiple of the alignment value (4)*1.
- The following three types of branch destination can be selected by specifying the suboptions of **-instalign4***2.
- No specification:Head of function and **case** and **default** labels of **switch** statement
  inmostloop: Head of each inmost loop, head of function, and **case** and **default** labels of **switch** statement
  loop: Head of each loop, head of function, and **case** and **default** labels of **switch** statement
- When this option is selected, the alignment value of the program section is changed from 1 to 4 (for **instalign4**) or 8 (for **instalign8**).
- This option aims to efficiently operate the instruction queues of the RX CPU and improve the speed of program execution by aligning the addresses of branch destination instructions.
  This option has specifications targeting the following usage.
- **instalign4**: When attempting to improve the speed of CPUs with a 32-bit instruction queue (mainly RX200 Series)

Notes 1. This is when the instruction size is equal to or smaller than the alignment value. If the instruction size is greater than the alignment value, alignment is performed only when the number of exceeding points is two or more.

2. Alignment is adjusted only for the branch destinations listed above; alignment of the other destinations is not adjusted. For example, when **loop** is selected, alignment of the head of a **loop** is adjusted but alignment is not adjusted at the branch destination of an **if** statement that is used in the loop but does not generate a loop.

-instalign8

[Format]

```
-instalign8[={loop|inmostloop}]
```

[Description]

- This option aligns instructions at branch destinations.
- When the **instalign8** option is specified, the instruction at the location address is aligned to the 8-byte boundary.
- Instruction alignment is performed only when the instruction at the specified location exceeds the address which is a multiple of the alignment value (8)*1.
- The following three types of branch destination can be selected by specifying the suboptions of **-instalign4** and **-instalign8***2.
- No specification:Head of function and **case** and **default** labels of **switch** statement
  inmostloop: Head of each inmost loop, head of function, and **case** and **default** labels of **switch** statement
  loop: Head of each loop, head of function, and **case** and **default** labels of **switch** statement
- When these options are selected, the alignment value of the program section is changed from 1 to 4 (for **instalign4**) or 8 (for **instalign8**).
- These options aim to efficiently operate the instruction queues of the RX CPU and improve the speed of program execution by aligning the addresses of branch destination instructions.
  This option has specifications targeting the following usage.
- **instalign8**: When attempting to improve the speed of CPUs with a 64-bit instruction queue (mainly RX600 Series)

Notes 1. This is when the instruction size is equal to or smaller than the alignment value. If the instruction size is greater than the alignment value, alignment is performed only when the number of exceeding points is two or more.

2. Alignment is adjusted only for the branch destinations listed above; alignment of the other destinations is not adjusted. For example, when **loop** is selected, alignment of the head of a loop is adjusted but alignment is not adjusted at the branch destination of an **if** statement that is used in the loop but does not generate a loop.

[Example]

- <C source file>

```
dlong a;
int f1(int num)
{
    return (num+1);
}
void f2(void)
{
    a = 0;
}
void f3(void)
{
}
```

- <Output code>

[When compiling with -instalign8 specified]

In the example shown below, the head of each function is aligned so that the instruction does not exceed the 8-byte boundary.

In 8-byte boundary alignment of instructions, the address will not be changed unless the target instruction exceeds the 8-byte boundary. Therefore, only the address of function f2 is actually aligned.

```
      .SECTION P,CODE,ALIGN=8
      .INSTALIGN 8
 _f1:                     ; Function f1, address = 0000H
      ADD     #01H,R1     ; 2 bytes
      RTS                 ; 1 byte
      .INSTALIGN 8
 _f2:                     ; Function f2, address =0008H
                          ; Note: Alignment is performed.
                          ; When a 6-byte instruction is placed at
                          ; 0003H, it exceeds the 8-byte boundary.
                          ; Thus, alignment is performed.
      MOV.L   #_a,R4      ; 6 bytes
      MOV.L   #0,[R4]     ; 3 bytes
      RTS                 ; 1 byte
      .INSTALIGN 8
 _f3:                     ; Function f3, address = 0012H
       RTS
      .END
```

-noinstalign

[Format]

```
-noinstalign
```

[Description]

- This option does not aligns instructions at branch destinations.

-nouse_div_inst

[Format]

```
-nouse_div_inst
```

[Description]

- This option generates code in which no DIV, DIVU, or FDIV instructions are used for division and modular division operations in the program.

[Remarks]

- This option calls the equivalent runtime functions instead of DIV, DIVU, or FDIV instructions. This may lower code efficiency in terms of required ROM capacity and speed of execution.

List Options

The following list options are available.

- -listfile
- -nolistfile
- -show

-listfile

[Format]

```
-listfile[={<file name>|<path name>}]
```

[Description]

- These options specify whether to output a source list file.
- When the **listfile** option is specified, a source list file is output. <file name> can also be specified.
- An existing folder can also be specified as <path name> instead of <file name>. In such a case, a source list file with the file extension **.lst** and the name of the source file being compiled or assembled is output to the folder selected as <path name>.

[Remarks]

- A linkage list cannot be output by this option. In order to output a linkage list, specify the **list** option of the optimizing linkage editor by using the **lnkcmd** option.
- Information output from the compiler is written to the source list. For the source list file format, refer to Assemble List File.
- When you use <path name>, create the folder in advance. If the folder specified as <path name> does not exist, the compiler will assume that <file name> is selected.

-nolistfile

[Format]

```
-nolistfile
```

[Description]

- When the **nolistfile** option is specified, no source list file is output.

-show

[Format]

```
-show=<sub>[,...]
      <sub>: { source | conditionals | definitions | expansions }
```

[Description]

- This option sets the source list file contents.
- The suboptions and specified contents are shown in the following table.

Table A-5. Suboption Specifications

Suboption	Description
source	Outputs the C/C++ source file.
conditionals	Outputs also the statements for which the specified condition is not satisfied in conditional assembly.
definitions	Outputs the information before **.DEFINE** replacement.
expansions	Outputs the assembler macro expansion statements.

[Remarks]

- This option is valid only when the **listfile** option has been specified.
- Information output from the compiler is written to the source list. For the source list file format, refer to Assemble List File.

Optimize Options

The following optimize options are available.

- -optimize
- -goptimize
- -speed
- -size
- -loop
- -inline
- -noinline
- -file_inline
- -case
- -volatile
- -novolatile
- -const_copy
- -const_div
- -noconst_div
- -library
- -scope
- -noscope
- -schedule
- -noschedule
- -map
- -nomap
- -approxdiv
- -enable_register
- -simple_float_conv
- -fpu
- -nofpu
- -alias
- -float_order
- -ip_optimize
- -merge_files
- -whole_program

-optimize

[Format]

```
-optimize = { 0 | 1 | 2 | max }
```

[Description]

- This option specifies the optimization level.
- When **optimize=0** is specified, the compiler does not optimize the program. Accordingly, the debugging information may be output with high precision and source-level debugging is made easier.
- When **optimize=1** is specified, the compiler partially optimizes the program by automatically allocating variables to registers, integrating the function exit blocks, integrating multiple instructions which can be integrated, etc. Accordingly, the code size may become smaller than when compiled with the **optimize=0** specification.
- When **optimize=2** is specified, the compiler performs overall optimization. However, the optimization contents to be performed slightly differ depending on whether the **size** option or **speed** option has been selected.
- When **optimize=max** is specified, the compiler performs optimization as much as possible. For example, the optimization scope is expanded to its maximum extent, and if the **speed** option is specified, loop expansion is possible on a large scale. Though the advantages of optimization can be expected, there may be side effects, such as longer compilation time, and if the **speed** option is specified, significantly increased code size.

[Remarks]

- If the default is not included in the description of an optimize option, this means that the default varies depending on the **optimize** option and **speed** or **size** option specifications. For details on the default, refer to the **speed** or **size** option.

-goptimize

[Format]

```
-goptimize
```

[Description]

- This option generates the additional information for inter-module optimization in the output file.
- At linkage, inter-module optimization is applied to files for which this option has been specified.

-speed

[Format]

```
-speed
```

[Description]

- When the **speed** option is specified, optimization will be performed with emphasis on execution performance.

[Remarks]

- When the **speed** option is specified, the following options are automatically specified based on the **optimize** option specification.
- When **optimize=max** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
speed	loop=8	inline=250	const_div	schedule	const_copy	noscope	map* nomap*	alias=ansi

Note The default is **map** when a C/C++ source program has been specified for input and **output=abs** or **output=mot** has been specified for output. For any other case, the default is **nomap**.

- When **optimize=2** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
speed	loop=2	inline=100	const_div	schedule	const_copy	scope	nomap	alias=noansi

- When **optimize=0** or **optimize=1** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
speed	loop=1	noinline	const_div	noschedule	noconst_copy	scope	nomap	alias=noansi

-size

[Format]

```
-size
```

[Description]

- When the **size** option is specified, optimization will be performed with emphasis on code size.

[Remarks]

- When the **size** option is specified, the following options are automatically specified based on the **optimize** option specification. Note however that if one of the following options is specified otherwise explicitly, that specified option becomes valid.
- When **optimize=max** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
size	loop=1	inline=0	noconst_div	schedule	const_ copy	noscope	map* nomap*	alias=ansi

Note The default is **map** when a C/C++ source program has been specified for input and **output=abs** or **output=mot** has been specified for output. For any other case, the default is **nomap**.

- When **optimize=2** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
size	loop=1	noinline	noconst_div	schedule	const_ copy	scope	nomap	alias=noansi

- When **optimize=0** or **optimize=1** is specified

	Loop Expansion	Inline Expansion	Converting Constant Division into Multiplication	Scheduling Instructions	Constant Propagation of const Qualified Variables	Dividing Optimizing Ranges	Optimizing External Variable Accesses	Optimization Considering the Type of the Data Indicated by the Pointer
size	loop=1	noinline	noconst_div	noschedule	noconst_copy	scope	nomap	alias=noansi

-loop

[Format]

```
-loop[=<numeric value>]
```

- [Default]
  The default for this option is **loop=2**.

[Description]

- This option specifies whether to optimize loop expansion.
- When the **loop** option is specified, the compiler expands loop statements (**for**, **while**, and **do-while**).
- The maximum expansion factor can be specified by <numeric value>. An integer from 1 to 32 can be specified for <numeric value>. If no <numeric value> is specified, 2 will be assumed.
- The default for this option is determined based on the **optimize** option and **speed** or **size** option specifications. For details, refer to the **speed** or **size** option.

[Remarks]

- This option is invalid when **optimize=0** or **optimize=1**.

-inline

[Format]

```
-inline[=<numeric value>]
```

- [Default]
  The default for this option is inline=100.

[Description]

- These options specify whether to automatically perform inline expansion of functions.
- A value from 0 to 65535 is specifiable as **<numeric value>**.
- When the **inline** option is specified, the compiler automatically performs inline expansion. However, inline expansion is not performed for the functions specified by **#pragma noinline**. The user is able to use **inline=<numeric value>**, to specify the allowed increase in the function's size due to the use of inline expansion. For example, when **inline=100** is specified, inline expansion will be performed until the function size has increased by 100% (size is doubled).
- The default for this option is determined based on the **optimize** option and **speed** or **size** option specifications. For details, refer to the **speed** or **size** option.

[Remarks]

- Inline expansion is attempted for all functions for which **#pragma inline** has been specified or with an **inline** specifier whether other options have been specified or not. To perform inline expansion for a function for certain, specify **#pragma inline** for the function. Even though this option has been selected or an **inline** specifier has been specified for the function, if the compiler judges that the efficiency is degraded by inline expansion, it will not perform it in some cases.

-noinline

[Format]

```
-noinline
```

[Description]

- When the **noinline** option is specified, automatic inline expansion is not performed.

[Remarks]

- Inline expansion is attempted for all functions for which **#pragma inline** has been specified or with an **inline** specifier whether other options have been specified or not.

-file_inline

[Format]

```
-file_inline=<file name>[,...]
```

- [Default]
  None

[Description]

- This option is not available in V.2.00. Any specification of this option will simply be ignored and will not lead to an error due to compatibility with former versions.

[Remarks]

- For C (C99) source files, **-merge_files** can be used instead of **-file_inline**. Add the file that was used with **-file_inline** (including the file path if **-file_inline_path** was used together with it) as one of the source files to be merged.
- There are some points to be noted regarding **-merge_files**. Refer to [Remarks] of the **-merge_files** option.

-case

[Format]

```
-case={ ifthen | table | auto }
```

- [Default]
  The default for this option is **case=auto**.

[Description]

- This option specifies the expansion method of the **switch** statement.
- When **case=ifthen** is specified, the **switch** statement is expanded using the **if_then** method, which repeats, for each **case** label, comparison between the value of the evaluation expression in the **switch** statement and the **case** label value. If they match, execution jumps to the statement of the **case** label. This method increases the object code size depending on the number of case labels in the **switch** statement.
- When **case=table** is specified, the **switch** statement is expanded by using the table method, where the **case** label jump destinations are stored in a branch table so that a jump to the statement of the **case** label that matches the expression for evaluation in the **switch** statement is made through a single access to the branch table. With this method, the size of the branch table increases with the number of **case** labels in the **switch** statement, but the performance in execution remains the same. The branch table is output to a section for areas holding **switch** statements for branch tables.
- When **case=auto** is specified, the compiler automatically selects the **if_then** method or table method.

[Remarks]

- The branch table created when **case=table** has been specified will be output to section **W** when the **nostuff** option is specified and will be output to section **W**, **W_2**, or **W_1** according to the size of the **switch** statement when the **nostuff** option is not specified.

-volatile

[Format]

```
-volatile
```

[Description]

- When **volatile** is specified, all external variables are handled as if they were **volatile** qualified. Accordingly, the access count and access order for external variables are exactly the same as those written in the C/C++ source file.

-novolatile

[Format]

```
-novolatile
```

[Description]

- When **novolatile** is specified, the external variables which are not **volatile** qualified are optimized. Accordingly, the access count and access order for external variables may differ from those written in the C/C++ source file.

-const_copy

[Format]

```
-const_copy
```

- [Default]
  The default for this option is **const_copy** when the **optimize=2** or **optimize=max** option has been specified.

[Description]

- When **const_copy** is specified, constant propagation is performed even for **const** qualified global variables.
- The default for this option is **const_copy** when the **optimize=2** or **optimize=max** option has been specified.

[Remarks]

- **const** qualified variables in a C++ source file cannot be controlled by this option (constant propagation is always performed).

-noconst_copy

[Format]

```
-noconst_copy
```

- [Default]
  The default for this option is **noconst_copy** when the **optimize=1** or **optimize=0** option has been specified.

[Description]

- When **noconst_copy** is specified, constant propagation is disabled for **const** qualified global variables.

[Remarks]

- **const** qualified variables in a C++ source file cannot be controlled by this option (constant propagation is always performed).

-const_div

[Format]

```
-const_div
```

- [Default]
  The default for this option is **const_div** when the **speed** option has been specified.

[Description]

- When **const_div** is specified, calculations for division and remainders of integer constants in the source file are converted into sequences of multiplication or bitwise operation (shift or bitwise AND operations) instructions.

[Remarks]

- Constant multiplication that can be performed through only shift operations and division and residue that can be performed through only bitwise AND operations cannot be controlled by the **const_div** option.

-noconst_div

[Format]

```
-noconst_div
```

- [Default]
  The default for this option is **noconst_div** when the **size** option has been specified.

[Description]

- When **noconst_div** is specified, the corresponding division and remainder instructions are used for calculating division and remainders of integer constants in the source file (except divisions and remainders of unsigned integers by powers of two).

[Remarks]

- Constant multiplication that can be performed through only shift operations and division and residue that can be performed through only bitwise AND operations cannot be controlled by the **noconst_div** option.

-library

[Format]

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

- [Default]
  The default for this option is **library=intrinsic**.

[Description]

- When **library=function** is specified, all library functions are called.
- When **library=intrinsic** is specified, instruction expansion is performed for **abs()**, **fabsf()**, and library functions which can use string manipulation instructions.

-scope

[Format]

```
-scope
```

- [Default]
  The default for this option is **scope** when the **optimize=max** option has been specified.

[Description]

- When the **scope** option is specified, the optimizing ranges of the large-size function are divided into many sections before compilation.
- Use this option at performance tuning because it affects the object performance depending on the program.

-noscope

[Format]

```
-noscope
```

- [Default]
  The default for this option is **noscope** when the **optimize=max** option has been specified.

[Description]

- When the **noscope** option is specified, the optimizing ranges are not divided before compilation. When the optimizing range is expanded, the object performance is generally improved although the compilation time is delayed. However, if registers are not sufficient, the object performance may be lowered. Use this option at performance tuning because it affects the object performance depending on the program.

-schedule

[Format]

```
-schedule
```

- [Default]
  The default for this option is **schedule** when the **optimize=2** or **optimize=max** option has been specified.

[Description]

- When the **schedule** option is specified, instructions are scheduled taking into consideration pipeline processing.

-noschedule

[Format]

```
-noschedule
```

- [Default]
  The default for this option is **noschedule** when the **optimize=1** or **optimize=0** option has been specified.

[Description]

- When the **noschedule** option is specified, instructions are not scheduled. Basically, processing is performed in the same order the instructions have been written in the C/C++ source file.

-map

[Format]

```
-map[= <file name>]
```

- [Default]
  The default for this option is map when the **optimize=max** option has been specified.

[Description]

- This option optimizes accesses to global variables.
- When the **map** option is specified, a base address is set by using an external symbol-allocation information file created by the optimizing linkage editor, and a code that uses addresses relative to the base address for accesses to global or static variables is generated.
- When accesses to external variables are to be optimized by the map option, how the **map** option is used differs according to the specification of the **output** option.
- [**output=abs** or **output=mot** is specified]
  Specify only **map** (not necessary when **optimize=max** is specified). Compilation and linkage are automatically performed twice, and a code in which the base address is set based on external symbol allocation information is generated.
- [**output=obj** is specified]
  Compile the source file once without specifying these options, create an external symbol-allocation information file by specifying **map=<file name>** at linkage by the optimizing linkage editor, and then compile the source file again by specifying **map=<file name>** in **ccrx**.

[Example]

- <C source file>

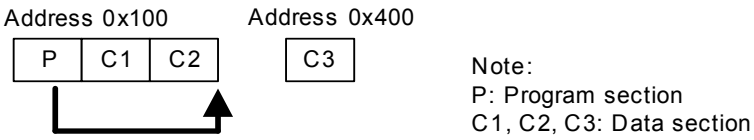
```
long A,B,C;
void func()
{
    A = 1;
    B = 2;
    C = 3;
}
```

- <Output code>

```
_func:
  MOV.L     #_A,R4     ; Sets the address of A as the base address.
  MOV.L     #1,[R4]
  MOV.L     #2,4[R4]   ; Accesses B using the address of A as the base.
  MOV.L     #3,8[R4]   ; Accesses C using the address of A as the base.
```

[Remarks]

- When the order of the definitions of global variables or static variables has been changed, a new external symbol-allocation information file must be created. If any option other than the **map** option in the previous compilation differs from the one in the current compilation, or if any contents of a function are changed, correct operation is not guaranteed. In such a case, a new external symbol-allocation information file must be created.
- This option is only valid for the compilation of C/C++ source programs. It does not apply to programs that have been compiled with the **output=src** specification or to programs written in assembly language.
- When the **map** option and **smap** option are specified simultaneously, the **map** option is valid.
- When continuous data sections are allocated after a program section, optimization of external variable accesses may be disabled or may not be performed sufficiently. For performing optimization to a maximum extent in a case in which multiple sections are allocated continuously, allocate the program section at the end. An example is shown below.



- In the above example, section **P** is allocated from address 0x100, sections **C1** and **C2** are allocated immediately after section **P**, and section **C3** is allocated from address 0x400. Since sections **C1** and **C2** are allocated continuously after section **P**, section **P** should be allocated behind section **C2**. Section **C3** is not involved because it is not allocated continuously.

-smap

[Format]

```
-smap
```

[Description]

- When the **smap** option is specified, a base address is set for global or static variables defined in the file to be compiled, and a code that uses addresses relative to the base address for accesses to those variables is generated.

[Example]

- <C source file>

```
long A,B,C;
void func()
{
    A = 1;
    B = 2;
    C = 3;
}
```

- <Output code>

```
_func:
  MOV.L     #_A,R4     ; Sets the address of A as the base address.
  MOV.L     #1,[R4]
  MOV.L     #2,4[R4]   ; Accesses B using the address of A as the base.
  MOV.L     #3,8[R4]   ; Accesses C using the address of A as the base.
```

[Remarks]

- This option is only valid for the compilation of C/C++ source programs. It does not apply to programs that have been compiled with the **output=src** specification or to programs written in assembly language.
- When the **map** option and **smap** option are specified simultaneously, the **map** option is valid.

-nomap

[Format]

```
-nomap
```

- [Default]
  The default for this option is **nomap** when the **optimize=0**, **optimize=1**, or **optimize=2** option has been specified.

[Description]

- When the **nomap** option is specified, accesses to external variables are not optimized.

[Example]

- <C source file>

```
long A,B,C;
void func()
{
    A = 1;
    B = 2;
    C = 3;
}
```

- <Output code>

```
_func:
     MOV.L     #_A,R4
     MOV.L     #1,[R4]
     MOV.L     #_B,R4
     MOV.L     #2,[R4]
     MOV.L     #_C,R4
     MOV.L     #3,[R4]
```

-approxdiv

[Format]

```
-approxdiv
```

- [Default]
  When this option is omitted, division of floating-point constants into multiplications of the corresponding reciprocals as constants is not performed.

[Description]

- This option converts divisions of floating-point constants into multiplications of the corresponding reciprocals as constants.
- To be specific, when there is an expression of (variable ÷ divisor) with the divisor being a constant, a code with the expression converted into (variable × reciprocal of divisor) will be generated.

[Remarks]

- When this option is specified, the execution performance of floating-point constant division will be improved. The precision and order of operations may, however, be changed, so take care on this point.

-enable_register

[Format]

```
-enable_register
```

[Description]

- This option is not available in V.2.00. Any specification of this option will simply be ignored and will not lead to an error due to compatibility with former versions.

-simple_float_conv

[Format]

```
-simple_float_conv
```

[Description]

- This option omits part of the type conversion processing for the floating type.
- When this option is selected, the generation code that performs type conversion of the next floating-point number changes.
  - Type conversion from 32-bit floating type to unsigned integer type
  - Type conversion from unsigned integer type to 32-bit floating type
  - Type conversion from integer type to 64-bit floating type via 32-bit floating type

[Example]

- <Type conversion from 32-bit floating type to unsigned integer type>

```
unsigned long func1(float f)
{
    return ((unsigned long)f);
}

When this option is not specified:
 _func1:
         FCMP   #4F000000H,R1
         BLT    L12
         FADD   #0CF800000H,R1
 L12:
         FTOI   R1,R1
         RTS
```

- <Type conversion from unsigned integer type to 32-bit floating type>

```
float func2(unsigned long u)
{
       return ((float)u);
}

When this option is not specified:
 _func2:
         BTST   #31,R1
         BEQ    L15
         SHLR   #1,R1,R14
         AND    #1,R1
         OR     R14,R1
         ITOF   R1,R1
         FADD   R1,R1
         BRA    L16
 L15:
         ITOF   R1,R1
 L16:
         RTS
```

- <Type conversion from integer type to 64-bit floating type via 32-bit floating type>

Note Does not apply when the dbl_size=8 specification is not valid.

```
double func3(long l)
 {
       return (double)(float)l;
 }
When this option is not specified:
_func3:
        ITOF    R1,R1
        BRA     __COM_CONVfd

When this option is specified:
        BRA     __COM_CONV32sd
```

[Remarks]

- When this option is specified, code performance of the relevant type conversion processing is improved. The conversion result may, however, differ from C/C++ language specifications, so take care on this point.
- This option is invalid when **optimize=0**.

-fpu

[Format]

```
-fpu
```

- [Default]
  The default for this option is **fpu** when RX600 is selected* as the CPU.

Note This means to be selected by either the cpu option or the environment variable CPU_RX.

[Description]

- When the **fpu** option is specified, a code using FPU instructions is generated.

[Remarks]

- For details of the FPU instructions, refer to the RX Family Software Manual.
- When RX200 is selected as the CPU, an error will occur if fpu is specified.

-nofpu

[Format]

```
-nofpu
```

- [Default]
  The default for this option is **fpu** when RX600 is selected* as the CPU.

Note This means to be selected by either the cpu option or the environment variable CPU_RX.

[Description]

- When the **nofpu** option is specified, a code not using FPU instructions is generated.

[Example]

-

[Remarks]

- For details of the FPU instructions, refer to the RX Family Software Manual.

-alias

[Format]

```
-alias = { noansi | ansi }
```

- [Default]
  The default for this option is alias=noansi.

[Description]

- This option selects whether to perform optimization with consideration for the type of the data indicated by the pointer.
- When alias=ansi is specified, based on the ANSI standard, optimization considering the type of the data indicated by the pointer is performed. Although the performance of object code is generally better than when alias=noansi is specified, the results of execution may differ according to whether alias=ansi or alias=noansi is specified.
- In the same way as in V. 1.00, ANSI-standard based optimization in consideration of the type of data indicated by pointers is not performed when alias=noansi is specified.

[Example]

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

- [When alias=noansi is specified]

Note The value of n is reloaded at (A) since it is regarded that there is a possibility of the value of n being rewritten by *ps = 2.

```
_func:
    MOV.L       #_n,R4
    MOV.L       #1,[R4]     ; n = 1;
    MOV.W       #2,[R1]     ; *ps = 2;
    MOV.L       [R4],R5     ; (A) n is reloaded
    MOV.L       #_x,R4
    MOV.L       R5,[R4]
    RTS
```

- [When alias=ansi is specified]

Note The value used in assignment at **n = 1** is reused at (B) because it is regarded that the value of **n** will not change at ***ps = 2** since ***ps** and **n** have different types.
(If the value of **n** is changed by ***ps = 2**, the result is also changed.)

```
_func:
    MOV.L       #_n,R4
    MOV.L       #1,[R4]     ; n = 1;
    MOV.W       #2,[R1]     ; *ps = 2;
    MOV.L       #_x,R4
    MOV.L       #1,[R4]     ; (B) Value used in assignment at n = 1 is reused
    RTS
```

[Remarks]

- When **optimize=0** or **optimize=1** is valid and the **alias** option is specified, the **alias=ansi** specification will be ignored and code will always be generated as if **alias=noansi** has been selected.

-float_order

[Format]

```
-float_order
```

- [Default]
  If this option is omitted, optimization of modification of the operation order in a floating-point expression is not performed.

[Description]

- This option is not available in V.2.00. Any specification of this option will simply be ignored and will not lead to an error due to compatibility with former versions.

-ip_optimize

[Format]

```
-ip_optimize
```

[Description]

- This option applies global optimization including
  - optimization that utilizes interprocedural alias analysis and
  - propagation of constant parameters and return values.

[Example]

1.

- <C source code>

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

- <Output assembly code without ip_optimize>

```
; -optimize=2 -size
__$func1:
  MOV.L #00000000H, [R1]
  MOV.L #00000001H, [R2]
  MOV.L [R1], R1
  RTS
_func2:
  MOV.L #_x,R1
  ADD #04H, R1, R2
  BRA __$func1
```

- <Output assembly code with ip_optimize>

```
; -optimize=2 -size
__$func1:
  MOV.L #00000000H, [R1]
  MOV.L #00000001H, [R2]
  MOV.L #00000000H, R1
  RTS
_func2:
  MOV.L #_x,R1
  ADD #04H, R1, R2
  BRA __$func1
```

2.

- <C source code>

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

- <Output assembly code without ip_optimize>

```
__$func:
  ADD R3, R1
  SUB R2, R1
  RTS
_func2:
  MOV.L #00000005H, R3
  MOV.L #00000004H, R2
  MOV.L #00000003H, R1
  BRA __$func
```

- <Output assembly code with ip_optimize>

```
__$func:
  MOV.L #00000004H, R1
  RTS
_func2:
  MOV.L #00000005H, R3
  MOV.L #00000004H, R2
  MOV.L #00000003H, R1
  BRA __$func
```

[Remarks]

- Inter-file optimization is also applied when this option is used with **merge_files**.

-merge_files

[Format]

```
-merge_files
```

[Description]

- This option allows the compiler to compile multiple source files and output the results to a single object file.
- The name of the object file is specified by the output option. If no name is specified, the filename will be that of the first source file plus a filename extension that corresponds to the selected output format.
- If **src** or **obj** is selected as the output format, the compiler also generates blank files that have the names of the other source files with the given filename extension attached.

[Example]

```
ccrx -merge_files -output=src=files.obj file1.c file2.c file3.c
```

files.obj is the object file. Blank files **file1.obj**, **file2.obj**, and **file3.obj** are also generated.

[Remarks]

- This option is invalid when only one source file is to be compiled or when the **output** option has been used to specify **prep** as the output format.
- Inter-file in-line expansion is applied when this option is used with the **inline** option.
- This option is not available for files to be compiled in C++ or EC++.
- The following restrictions apply to programs that include static functions or static variables.
  - If you wish to use the [Watch] window of the debugger to view a static variable that has the same name as a variable in another file, specify the variable name as well as the filename. The debugger cannot identify the variable without a filename.
  - When two or more files contain static variables with the same name and **rlink** is used to overlay sections to which the files belong, the debugger's facility to display overlay sections taking precedence over other sections is not available.
  - The names of static variables and static functions written in the link map file (.map) are those converted by the compiler (i.e., not original ones).
- Any differences (e.g. type specifier) in declarations of the same variable may lead to an error in compilation.

-whole_program

[Format]

```
-whole_program
```

[Description]

- This option makes the compiler perform optimization on the assumption that all source files have been input.

[Remarks]

- Specifying this option also makes the **ip_optimize** option effective, and if multiple source files are input, the **merge_files** option is also effective.
- When this option is specified, compilation is on the assumption that the conditions listed below are satisfied. Correct operation is not guaranteed otherwise.
  - Values and addresses of **extern** variables defined in the target source files will not be modified or referred to by other files.
  - Functions within the target source file will not be called from within other files, although functions in other files can be called from within the target source files.

[Example]

```
[wp.c]
extern void g(void);
int func(void)
{
    static int a = 0;
    a++;      // (1) Write a value to a.
    g();      // (2) Call g().
    return a; // (3) Call a.
}


[Without whole_program]
The compiler assumes that (2) will change the value of a since function g() may call
function func(), and generates a code to read the value of a in (3).
_func:
        PUSH.L R6
        MOV.L  #__$a$1,R6
        MOV.L  [R6],R14
        ADD    #1,R14
        MOV.L  R14,[R6]     ; (1)
        BSR    _g           ; (2)
        MOV.L  [R6],R1      ; (3)
        RTSD   #4,R6-R6

[With whole_program]
The compiler assumes that function g() will not call function func() and thus (2) will not
change the value of a. As a result, the compiler does not read the value of a in (3) and
instead generates a code to use the value written to a in (1).
_func:
        PUSH.L R6
        MOV.L  #__$a$1,R14
        MOV.L  [R14],R6
        ADD    #1,R6
        MOV.L  R6,[R14]     ; (1)
        BSR    _g           ; (2)
        MOV.L  R6,R1        ; (3)
        RTSD   #4,R6-R6
```

Microcontroller Options

The following microcontroller options are available.

- -cpu
- -endian
- -round
- -denormalize
- -dbl_size
- -int_to_short
- -signed_char
- -unsigned_char
- -signed_bitfield
- -unsigned_bitfield
- -auto_enum
- -bit_order
- -pack
- -unpack
- -exception
- -noexception
- -rtti
- -fint_register
- -branch
- -base
- -patch
- -pic
- -pid
- -nouse_pid_register
- -save_acc

-cpu

[Format]

```
-cpu={ rx600 | rx200 }
```

- [Default]
  The default for this option is determined based on the environment variable **RX_CPU**.

[Description]

- This option specifies the microcontroller type for the instruction code to be generated.
- When **cpu=rx600** is specified, an instruction code for the RX600 Series is generated.
- When **cpu=rx200** is specified, an instruction code for the RX200 Series is generated.

[Remarks]

- When **cpu=rx200** is specified, the **nofpu** option is automatically selected.
- **cpu=rx200** and the **fpu** option cannot be specified at the same time.
- When **cpu=rx600** is specified while neither the **nofpu** option nor the **fpu** option has been specified, the **fpu** option is automatically selected.

-endian

[Format]

```
-endian={ big  |  little }
```

- [Default]
  The default for this option is **endian=little**.

[Description]

- When **endian=big** is specified, data bytes are arranged in big endian.
- When **endian=little** is specified, data bytes are arranged in little endian.
- The endian type can also be specified by the **#pragma endian** extension. If both this option and a **#pragma** extension are specified, the **#pragma** specification takes priority.

-round

[Format]

```
-round={ zero  |  nearest }
```

- [Default]
  The default for this option is **round=nearest**.

[Description]

- This option specifies the rounding method for floating-point constant operations.
- When **round=zero** is specified, values are rounded to zero.
- When **round=nearest** is specified, values are rounded to the nearest value.

[Remarks]

- This option does not affect the method of rounding for floating-point operations during program execution.
- The default selection of this option does not affect the selection of the **fpu** and **nofpu** options.

-denormalize

[Format]

```
-denormalize={ off  |  on }
```

- [Default]
  The default for this option is **denormalize=off**.

[Description]

- This option specifies the operation when denormalized numbers are used to describe floating-point constants.
- When **denormalize=off** is specified, denormalized numbers are handled as zero.
- When **denormalize=on** is specified, denormalized numbers are handled as they are.

[Remarks]

- This option does not affect the handling of denormalized numbers in floating-point operations during program execution.
- This option is not automatically enabled by the selection of the **fpu** and **nofpu** options.

-dbl_size

[Format]

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

- [Default]
  The default for this option is **dbl_size=4**.

[Description]

- This option specifies the precision of the **double** type and **long double** type.
- When **dbl_size=4** is specified, the **double** type and **long double** type are handled as the single-precision floating type (4 bytes).
- When **dbl_size=8** is specified, the **double** type and **long double** type are handled as the double-precision floating type (8 bytes).

[Remarks]

- When **dbl_size=4** is selected, among the standard functions, the **mathf.h** and **math.h** functions having the same specifications as each other (e.g., **sqrtf** and **sqrt**) are integrated to configure a standard library. Because of this, phenomena, such as the following example will occur when **dbl_size=4** is selected. When the RX simulator or emulator traces (single-step execution) the calling of **sqrtf** which is a **mathf.h** header function, it appears as if not **sqrtf** but **sqrt**, which is a **math.h** header function with the same specifications, has been called.

-int_to_short

[Format]

```
-int_to_short
```

- [Default]
  Before compilation, the **int** type is not replaced with the **short** type and the **unsigned int** type is not replaced with the **unsigned short** type in the source file.

[Description]

- Before compilation, the **int** type is replaced with the **short** type and the **unsigned int** type is replaced with the **unsigned short** type in the source file.

[Remarks]

- **INT_MAX**, **INT_MIN**, and **UINT_MAX** of **limits.h** are not converted by this option.
- This option is invalid during C++ and EC++ program compilation. If an external name of a C program may be referred to by a C++, EC++ program, message W0523041 will be output for the external name.
- When the **int_to_short** option is specified and a file including a C standard header is compiled as C++ or EC++, the compiler may show the W0523041 message. In this case, simply ignore the message because it does not indicate a problem.
- Data that are shared between C and C++ (EC++) programs must be declared as the **long** or **short** type rather than as the **int** type.

-signed_char

[Format]

```
-signed_char
```

- [Default]
  The default for this option is **unsigned_char**.

[Description]

- When **signed_char** is specified, the value is handled as the **signed char** type.

[Remarks]

- The bit-field members of the **char** type are not controlled by this option; control them using the **signed_bitfield** and **unsigned_bitfield** options.

-unsigned_char

[Format]

```
-unsigned_char
```

- [Default]
  The default for this option is **unsigned_char**.

[Description]

- When **unsigned_char** is specified, the value is handled as the **unsigned char** type.

[Remarks]

- The bit-field members of the **char** type are not controlled by this option; control them using the **signed_bitfield** and **unsigned_bitfield** options.

-signed_bitfield

[Format]

```
-signed_bitfield
```

- [Default]
  When **signed_bitfield** is omitted, the value is handled as **unsigned**.

[Description]

- When **signed_bitfield** is specified, the value is handled as **signed**.

-unsigned_bitfield

[Format]

```
-unsigned_bitfield
```

- [Default]
  When **unsigned_bitfield** is omitted, the value is handled as **unsigned**.

[Description]

- When **unsigned_bitfield** is specified, the value is handled as **unsigned**.

-auto_enum

[Format]

```
-auto_enum
```

- [Default]
  The default for this option is to process the enumeration type size as the **signed long** type.

[Description]

- This option processes the enumerated data qualified by **enum** as the minimum data type with which the enumeration value can fit in.
- The possible enumeration values correspond to the data types as shown in the following table.

Table A-6. Correspondences between Possible Enumeration Values and Data Types

Enumerator		Data Type
Minimum Value	Maximum Value	
−128	127	signed char
0	255	unsigned char
−32768	32767	signed short
0	65535	unsigned short
Other than above	Other than above	signed long

-bit_order

[Format]

```
-bit_order = { left | right }
```

- [Default]
  The default for this option is **bit_order=right**.

[Description]

- This option specifies the order of bit-field members.
- When **bit_order=left** is specified, members are allocated from the upper bit.
- When **bit_order=right** is specified, members are allocated from the lower bit.
- The order of bit-field members can also be specified by the **#pragma bit_order** extension. If both this option and a **#pragma** extension are specified, the **#pragma** specification takes priority.

-pack

[Format]

```
-pack
```

- [Default]
  The boundary alignment value for structures and classes equals the maximum boundary alignment value for members.

[Description]

- This option specifies the boundary alignment value for structure members and class members.
- The boundary alignment value for structure members can also be specified by the **#pragma pack** extension. If both this option and a **#pragma** extension are specified, the **#pragma** specification takes priority. The boundary alignment value for structures and classes equals the maximum boundary alignment value for members.

[Remarks]

- The boundary alignment values for structure members and class members when these options are specified are shown in the following table.

Table A-7. Boundary Alignment Values for Structure Members and Class Members When the pack Option is Specified

Member Type	pack	Not Specified
(signed) char	1	1
(unsigned) short	1	2
(unsigned) int[Note], (unsigned) long, (unsigned) long long, floating type, and pointer type	1	4

Note Becomes the same as **short** when the **int_to_short** option is specified.

-unpack

[Format]

```
-unpack
```

- [Default]
  The boundary alignment value for structures and classes equals the maximum boundary alignment value for members.

[Description]

- This option specifies the boundary alignment value for structure members and class members.
- The boundary alignment value for structures and classes equals the maximum boundary alignment value for members.

[Remarks]

- The boundary alignment values for structure members and class members when these options are specified are shown in the following table.

Table A-8. Boundary Alignment Values for Structure Members and Class Members When the unpack Option is Specified

Member Type	unpack	Not Specified
(signed) char	1	1
(unsigned) short	2	2
(unsigned) int[Note], (unsigned) long, (unsigned) long long, floating type, and pointer type	4	4

Note Becomes the same as **short** when the **int_to_short** option is specified.

-exception

[Format]

```
-exception
```

- [Default]
  The C++ exceptional handling function (**try**, **catch**, **throw**) is disabled.

[Description]

- The C++ exceptional handling function (**try**, **catch**, **throw**) is enabled.
- The code performance may be lowered.

[Remarks]

- In order to use the C++ exceptional handling function among files, perform the following:
  - Specify **rtti=on**.
  - Do not specify the **noprelink** option in the optimizing linkage editor.
- The **exception** option can be specified only at C++ compilation. The **exception** option is ignored when **lang=cpp** has not been specified and the input file extension is **.c** or **.p**.

-noexception

[Format]

```
-noexception
```

- [Default]
  The C++ exceptional handling function (**try**, **catch**, **throw**) is disabled.

[Description]

- The C++ exceptional handling function (**try**, **catch**, **throw**) is disabled.

[Remarks]

- In order to use the C++ exceptional handling function among files, perform the following:
  - Specify **rtti=on**.
  - Do not specify the **noprelink** option in the optimizing linkage editor.
- The **noexception** option can be specified only at C++ compilation. The **noexception** option cannot be specified when **lang=cpp** has not been specified and the input file extension is **.c** or **.p**. If specified, an error will occur.

-rtti

[Format]

```
-rtti={ on | off }
```

- [Default]
  The default for this option is **rtti=off**.

[Description]

- This option enables or disables runtime type information.
- When **rtti=on** is specified, **dynamic_cast** and **typeid** are enabled.
- When **rtti=off** is specified, **dynamic_cast** and **typeid** are disabled.

[Remarks]

- Do not define relocatable files (**.obj**) that were created by this option in a library, and do not output files in the relocatable format (**.rel**) through the optimizing linkage editor. A symbol double definition error or symbol undefined error may occur.
- **rtti=on** can be specified only at C++ compilation. **rtti=on** is ignored when **lang=cpp** has not been specified and the input file extension is **.c** or **.p**.

-fint_register

[Format]

```
-fint_register = {0 | 1 | 2 | 3 | 4 }
```

- [Default]
  The default for this option is **fint_register=0**.

[Description]

- This option specifies the general registers which are to be used only in fast interrupt functions (functions that have the fast interrupt setting (**fint**) in their interrupt specification defined by #**pragma interrupt**). The specified registers cannot be used in functions other than the fast interrupt functions. Since the general registers specified by this option can be used without being saved or restored in fast interrupt functions, the execution speed of fast interrupt functions will most likely be improved. Then again, since the number of usable general registers in other functions is reduced, the efficiency of register allocation in the entire program is degraded.
- The options correspond to the registers as shown in the following table.

Table A-9. Correspondences between Options and Registers

Option	Registers for Fast Interrupts Only
fint_register=0	None
fint_register=1	R13
fint_register=2	R12, R13
fint_register=3	R11, R12, R13
fint_register=4	R10, R11, R12, R13

[Remarks]

- Correct operation is not guaranteed when a register specified by this option is used in a function other than the fast interrupt functions. If a register specified by this option has been specified by the **base** option, an error will occur.

-branch

[Format]

```
-branch = { 16 | 24 | 32 }
```

- [Default]
  The default for this option is **branch=24**.

[Description]

- This option specifies the branch width.
- When **branch=16** is specified, the program is compiled with a branch width within 16 bits.
- When **branch=24** is specified, the program is compiled with a branch width within 24 bits.
- When **branch=32** is specified, the branch width is not specified.

-base

[Format]

```
-base = { rom=<register>
        | ram=<register>
        | <address value> = <register>}
                            <register>:= {R8 to R13}
```

[Description]

- This option specifies the general register used as a fixed base address throughout the program.
- When **base=rom=<register A>** is specified, accesses to **const** variables are performed relative to the specified register A. Note that the total size of the constant area section must be within 64 Kbytes to 256 Kbytes*.
- When **base=ram=<register B>** is specified, accesses to initialized variables and uninitialized variables are performed relative to the specified register B. Note that the total RAM data size must be within 64 Kbytes to 256 Kbytes*.
- When **<address value>=<register C>** is specified, accesses to an area within 64Kbytes to 256 bytes from the address value, among the areas whose addresses are already determined at the time of compilation, are performed relative to the specified register C.

Note This value is in the range from 64 to 256 Kbytes and depends on the total size of variables to be accessed.

[Remarks]

- The same register cannot be specified for different areas.
- Only a single register can be specified for each area. If a register specified by the **fint_register** option is specified by this option, an error will occur.
- When the **pid** option is selected, **base=rom=<register>** cannot be selected. If selected, message W0523039 is output as a warning and the selection of **base=rom=<register>** is disabled.

-patch

[Format]

```
-patch = { rx610 }
```

[Description]

- This option is used to avoid a problem specific to the CPU type.
- When **-patch=rx610** is specified, the **MVTIPL** instruction which causes a problem in the RX610 Group is not used in the generated code. Unless **-patch=rx610** is specified, the code generated in response to the call by the intrinsic function **set_ipl** will contain the **MVTIPL** instruction.

-pic

[Format]

```
-pic
```

- [Default]
  This option does not generate code with the program section as PIC (position independent code).

[Description]

- This option generates code with the program section as PIC (position independent code).
- In PIC, all function calls are performed with BSR or BRA instructions. When acquiring the address of a function, a relative address from the PC should be used. This allows PIC to be located at a desired address after linkage.

[Example]

- Calling a function (only for **branch=32**)

```
void func()
{
        sub();
}
```

```
[Without -pic]
_func:
            MOV.L       #_sub,R14
            JMP         R14
[With -pic]
_func:
            MOV.L       #_sub-L11,R14
L11:
            BRA         R14
```

- Acquiring a function address

```
void func1(void);
void (*f_ptr)(void);
void func2(void)
{
        f_ptr = func1;
}
```

```
[Without -pic]
_func2:
            MOV.L       #_f_ptr,R4
            MOV.L       #_func1,[R4]
            RTS
[With -pic]
_func2:
         MOV.L       #_f_ptr,R4
L11:
         MVFC        PC,R14
         ADD         #_func1-L11,R14
         MOV.L       R14,[R4]
         RTS
```

[Remarks]

- In C++ or EC++ compilation, the **pic** option cannot be selected. If selected, message W0523039 is output as a warning and the selection of the **pic** option is disabled.
- The address of a function which is PIC should not be used in the initialization expression used for static initialization. If used, error E0523026 will occur.
- <Example of using a PIC address for static initialization>

```
void pic_func1(void), pic_func2(int), pic_func3(int); /* Becomes PIC */
void (*fptr1_for_pic) = pic_func1; /* Uses PIC address in static initialization: Error */
struct PIC_funcs{ int code; void (*fptr)(int); };
struct PIC_funcs pic_funcs[] = {
    { 2, pic_func2 },              /* Uses PIC address in static initialization: Error */
    { 3, pic_func3 },              /* Uses PIC address in static initialization: Error */
};
```

- When creating a code for startup of the application program using the PIC function, refer to Application Startup, instead of Startup.
- For the PIC function, also refer to Usage of PIC/PID Function.

-pid

[Format]

```
-pid[={ 16 | 32 }]
```

- [Default]
  The constant area sections **C**, **C_2**, and **C_1**, the literal section **L**, and the **switch** statement branch table sections **W**, **W_2**, and **W_1** are not handled as PID (position independent data).

[Description]

- The constant area sections **C**, **C_2**, and **C_1**, the literal section **L**, and the **switch** statement branch table sections **W**, **W_2**, and **W_1** are handled as PID (position independent data).
- PID can be accessed through a relative address from the PID register. This allows PID to be located at a desired address after linkage.
- A single general register is used to implement the PID function.
- <PID register>
  - Based on the rules in the following table, one register from among R9 to R13 is selected according to the specification of the **fint_register** option. If the **fint_register** option is not specified, R13 is selected.

Table A-10. Correspondences between fint_register Options and PID Registers

fint_register Option	PID Register
No **fint_register** specification	R13
fint_register = 0	
fint_register = 1	R12
fint_register = 2	R11
fint_register = 3	R10
fint_register = 4	R9

  - The PID register can be used only for the purpose of PID access.
- <Parameters>
  - The parameter selects the maximum bit width of the offset when accessing the constant area section from the PID register as 16 bits or 32 bits.
  - The default for this option when the offset width is omitted is **pid=16**. When **pid=16** is specified, the size of the constant area section that can be accessed by the PID register is limited to 64 Kbytes to 256 Kbytes (varies depending on the access width). When **pid=32** is specified, there is no limitation of the size of the constant area section that can be accessed by the PID register, but the size of the code accessing PID is increased.
  - Note that when **pid=32** and the map option with valid external symbol-allocation information are specified at the same time, the allocation information causes code the same as if **pid=16** was specified to be generated if access by the PID register is possible.

[Examples]

- Accessing an externally referenced symbol that is **const** qualified

```
extern const int pid;
int work;
void func1()
{
        work = pid;
}
```

```
[Without -pid]
_func1:
            MOV.L       #_pid,R4
            MOV.L       [R4],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
[With -pid=16] (only when the PID register is R13)
_func1:
            MOV.L       _pid-__PID_TOP:16[R13],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
[With -pid=32] (only when the PID register is R13)
_func1:
            ADD         #(_pid-__PID_TOP),R13,R6
            MOV.L       [R6],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
```

- Acquiring the address of an externally defined symbol that is **const** qualified

```
extern const int pid = 1000;
const int *ptr;
void func2()
{
        ptr = &pid;
}
```

```
[Without -pid]
_func2:
            MOV.L       #_ptr,R4
            MOV.L       #_pid,[R4]
            RTS
[With -pid] (only when the PID register is R13)
_func2:
            ADD         #(_pid-__PID_TOP),R13,R5
            MOV.L       #_ptr,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
```

[Remarks]

- The address of an area which is PID should not be used in the initialization expression used for static initialization. If used, error E0523027 will occur.
- <Example of using a PID address for static initialization>

```
extern const int pid_data1;          /* Becomes PID */
const int *ptr1_for_pid = &pid_data1;/* Uses PID address in static initialization: Error */
const int pid_data4[] = {1,2,3,4};   /* Becomes PID */
const int *ptr2_for_pid = pid_data4; /* Uses PID address in static initialization: Error */
```

- When creating a code for startup of the application program using the PID function, refer to Application Startup, instead of Startup.
- When the **pid** option is selected, the same external variables in different files all have to be **const** qualified. This is because the **pid** option is used to specify **const** qualified variables as PID. The **pid** option (PID function) should not be used when there may be an external variable that is not **const** qualified.
- If the **map=<file name>** option is enabled while the **pid** option is selected, warning W0530809 may be output when there is an externally referenced variable that is not **const** qualified but used in different files as the same external variable. In the case, the displayed variable is handled as PID.
- In C++ or EC++ compilation, the **pid** option cannot be selected. If selected, message W0523039 is output as a warning and the selection of the **pid** option is disabled.
- When the **pid** option is selected, **base=rom=<register>** cannot be selected. If selected, message W0523039 is output as a warning and the selection of **base=rom=<register>** is disabled.
- If a PID register selected by the **pid** option is also specified by the **base** option, warning W0511149 will occur.
- If the **pid** option and **nouse_pid_register** option are selected simultaneously, error E0511150 will occur.
- For details of the application and PID function, refer to Usage of PIC/PID Function.

-nouse_pid_register

[Format]

```
-nouse_pid_register
```

[Description]

- When this option is specified, the generated code does not use the PID register.
- Selection of the PID register according to the settings of the **fint_register** option is based on the same rule as for the **pid** option.
- A master program called by an application program in which the PID function is enabled needs to be compiled with this option. At this time, if the **fint_register** option is selected in the application program, the same parameter **fint_register** should also be selected in the master program.

[Remarks]

- If the **nouse_pid_register** option and **pid** option are selected simultaneously, error E0511150 will occur.
- A register selected as the PID register also being specified for the **base** option leads to warning W0511149.
- For details of the PID function, refer to Usage of PIC/PID Function.

-save_acc

[Format]

-save_acc

- [Default]
  When this option is omitted, it does not generate the saved and restored code of the accumulator (**ACC**) for interrupt functions.

[Description]

- This option generates the saved and restored code of the accumulator (**ACC**) for interrupt functions.

[Remarks]

- The generated saved and restored code is the same code generated when **acc** is selected in **#pragma interrupt**. For the actual saved and restored code, refer to the description of **acc** and **no_acc** in **#pragma interrupt** of #pragma Extension Specifiers and Keywords.

Assemble and Linkage Options

The following assemble and linkage options are available.

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

-asmcmd

[Format]

```
-asmcmd=<file name>
```

[Description]

- This option specifies the assembler options to pass to **asrx** with a subcommand file.

[Example]

```
ccrx -cpu=rx600 -asmcmd=file.sub sample.c
```

- The above description has the same meaning as the following two command lines:

```
ccrx -cpu=rx600 -output=src sample.c
asrx -cpu=rx600 -subcommand=file.sub sample.src
```

[Remarks]

- If this option is specified for more than one time, all specified subcommand files are valid.

-lnkcmd

[Format]

```
-lnkcmd=<file name>
```

[Description]

- This option specifies the linkage options to pass to **rlink** with a subcommand file.

[Example]

```
ccrx -cpu=rx600 -output=abs=tp.abs -lnkcmd=file.sub tp1.c tp2.c
```

- The above description has the same meaning as the following three command lines:

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c]
asrx -cpu=rx600 tp1.src tp2.src
rlink -subcommand=file.sub -form=abs -output=tp tp1.obj tp2.obj
```

[Remarks]

- If this option is specified for more than one time, all specified subcommand files are valid.

-asmopt

[Format]

```
-asmopt=["]<assembler option>["]
```

[Description]

- This option specifies the assembler options to pass to **asrx** with a string.
- Multiple options can be specified by enclosing them with double-quote marks (").

[Example]

```
ccrx -cpu=rx600 -asmopt="-chkpm" sample.c
```

- The above description has the same meaning as the following two command lines:

```
ccrx -cpu=rx600 -output=src sample.c
asrx -cpu=rx600 -chkpm sample.src
```

[Remarks]

- If this option is specified for more than one time, all specified assembler options are valid.

-lnkopt

[Format]

```
-lnkopt=["]<linkage option>["]
```

[Description]

- This option specifies the linkage options to pass to **rlink** with a string.
- Multiple options can be specified by enclosing them with double-quote marks (").

[Example]

```
ccrx -cpu=rx600 -output=abs=tp.abs -lnkopt="-start=P,C,D/100,B/8000" tp1.c tp2.c
```

- The above description has the same meaning as the following three command lines:

```
ccrx -cpu=rx600 -output=src tp1.c tp2.c
asrx -cpu=rx600 tp1.src tp2.src
rlink -start=P,C,D/100,B/8000 -form=abs -output=tp tp1.obj tp2.obj
```

[Remarks]

- If this option is specified for more than one time, all specified linkage options are valid.

Other Options

The following other options are available.

- -logo
- -nologo
- -euc
- -sjis
- -latin1
- -utf8
- -big5
- -gb2312
- -outcode
- -subcommand

-logo

[Format]

```
-logo
```

- [Default]
  The copyright notice is output.

[Description]

- The copyright notice is output.

-nologo

[Format]

```
-nologo
```

- [Default]
  The copyright notice is output.

[Description]

- When the **nologo** option is specified, output of the copyright notice is disabled.

-euc

[Format]

```
-euc
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **SJIS** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **euc** code.

-sjis

[Format]

```
-sjis
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **SJIS** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **SJIS** code.

-latin1

[Format]

```
-latin1
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **SJIS** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **latin1** code.

-utf8

[Format]

```
-utf8
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **SJIS** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **utf8** code.

[Remarks]

- The **utf8** option is valid only when the **lang=c99** option has been specified.

-big5

[Format]

```
-big5
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **BIG5** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **big5** code.

[Remarks]

- When **big5** is specified, the same character coding must be selected for the **outcode** option.

-gb2312

[Format]

```
-gb2312
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **GB2312** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **gb2312** code.

[Remarks]

- When **gb2312** is specified, the same character coding must be selected for the **outcode** option.

-outcode

[Format]

```
-outcode = { euc | sjis | latin1 | utf8 | big5 | gb2312 }
```

- [Default]
  The default for this option is **outcode=sjis**.

[Description]

- This option specifies the character code to output characters in strings and character constants.
- The options correspond to the character codes as shown in the following table.

Table A-11. Correspondences between Options and Character Codes (outcode)

Option	Character Code
euc	EUC code
sjis	SJIS code
latin1	ISO-Latin1 code
utf8	UTF-8 code
big5	Big5 code
gb2312	GB2312 code

[Remarks]

- The **utf8** option is valid only when the **lang=c99** option has been specified.
- When **outcode=big5** or **outcode=gb2312**, the **big5** or **gb2312** option must also be specified.

-subcommand

[Format]

```
-subcommand=<subcommand file name>
```

[Description]

- When the **subcommand** option is specified, the compiler options specified in a subcommand file are used at compiler startup. Specify options in a subcommand file in the same format as in the command line.

[Remarks]

- If this option is specified for more than one time, all specified subcommand files are valid.

(2) Assembler Command Options

Classification	Option	Description
Source Options	-include	Specifies the names of folders that hold include files.
	-define	Specifies macro definitions.
	-chkpm	Checks for a privileged instruction.
	-chkfpu	Checks for a floating-point operation instruction.
	-chkdsp	Checks for a DSP instruction.
Object Options	-output	Specifies the relocatable file name.
	-debug	Debugging information is output to the object files.
	-nodebug	Debugging information is not output to the object files.
	-goptimize	Outputs additional information for inter-module optimization.
List Options	-listfile	An assembler list file is output.
	-nolistfile	An assembler list file is not output.
	-show	Specifies the contents of the source list file.
Microcontroller Options	-cpu	Selects the microcontroller type.
	-endian	Selects the endian type.
	-fint_register	Selects a general register for exclusive use with the fast interrupt function.
	-base	Specifies the base registers for ROM and RAM.
	-patch	Selects avoidance or non-avoidance of a problem specific to the CPU type.
	-pic	Enables the PIC function.
	-pid	Enables the PID function.
	-nouse_pid_register	The PID register is not used in code generation.
Other Options	-logo	Selects the output of copyright information.
	-nologo	Selects the non-output of copyright information.
	-subcommand	Specifies a file for including command options.
	-euc	The character codes of input programs are interpreted as EUC codes.
	-sjis	The character codes of input programs are interpreted as SJIS codes.
	-latin1	The character codes of input programs are interpreted as ISO-Latin1 codes.
	-big5	The character codes of input programs are interpreted as BIG5 codes.
	-gb2312	The character codes of input programs are interpreted as GB2312 codes.

Source Options

The following source options are available.

- -include
- -define
- -chkpm
- -chkfpu
- -chkdsp

-include

[Format]

```
-include=<path name>[,...]
```

- [Default]
  The include file is searched for in the order of the current folder and the folders specified by environment variable **INC_RXA**.

[Description]

- This option specifies the name of the path to the folder that stores the include file.
- Multiple path names can be specified by separating them with a comma (,).
- The include file is searched for in the order of the current folder, the folders specified by the **include** option, and the folders specified by environment variable **INC_RXA**.

[Example]

- Folders **c:\usr\inc** and **c:\usr\rxc** are searched for the include file.

```
asrx -include=c:\usr\inc,c:\usr\rxc test.src
```

-define

[Format]

```
-define=<sub>[,...]
        <sub>: <macro name> = <string>
```

[Description]

- This option replaces the macro name with the specified string.
  (This provides the same function as writing the **.DEFINE** directive at the beginning of the source file.)

[Remarks]

- **.DEFINE** takes priority over the **define** option if both are specified.

-chkpm

[Format]

```
-chkpm
```

[Description]

- This option outputs warning W0551011 when a privileged instruction is used in the source file.

[Remarks]

- For details of the privileged instructions, refer to the RX Family Software Manual.

-chkfpu

[Format]

```
-chkfpu
```

[Description]

- This option outputs warning W0551012 when a floating-point operation instruction is used in the source file.

[Remarks]

- For details of the floating-point operation instructions, refer to the RX Family Software Manual.

-chkdsp

[Format]

```
-chkdsp
```

[Description]

- This option outputs warning W0551013 when a DSP instruction is used in the source file.

[Remarks]

- For details of the DSP instructions, refer to the RX Family Software Manual.

Object Options

The following object options are available.

- -output
- -debug
- -nodebug
- -goptimize

-output

[Format]

```
-output=<output file name>
```

- [Default]
  This option outputs a relocatable file having the same name as that of the source file with extension **.obj**.

[Description]

- When the specified output file name does not have an extension, the file name appended with extension **.obj** is used for the output relocatable file name. When it has an extension, the extension is replaced with **.obj**.

-debug

[Format]

```
-debug
```

- [Default]
  If this option is not specified, no debugging information is output to the relocatable file.

[Description]

- When the **debug** option is specified, debugging information is output to the relocatable file.

-nodebug

[Format]

```
-nodebug
```

- [Default]
  If this option is not specified, no debugging information is output to the relocatable file.

[Description]

- When the **nodebug** option is specified, no debugging information is output to the relocatable file.

-goptimize

[Format]

```
-goptimize
```

- [Default]
  If this option is not specified, additional information for the inter-module optimization is not output.

[Description]

- This option outputs the additional information for the inter-module optimization.
- At linkage, inter-module optimization is applied to the file specified with this option.

List Options

The following list options are available.

- -listfile
- -nolistfile
- -show

-listfile

[Format]

```
-listfile[=<file name>]
```

- [Default]
  If this option is not specified, no assemble list file is output.

[Description]

- When the **listfile** option is specified, an assemble list file is output. The name of the file can also be specified.
- <file name> should be specified according to the rules described in the Naming Files section.
- If <file name> is not specified in the **listfile** option, the source file name with the extension replaced with **.lst** is used as the source list file name.

-nolistfile

[Format]

```
-nolistfile
```

- [Default]
  If this option is not specified, no assemble list file is output.

[Description]

- When the **nolistfile** option is specified, no assemble list file is output.

-show

[Format]

```
-show=<sub>[,...]
      <sub>: { conditionals | definitions | expansions }
```

[Description]

- This option specifies the contents of the list file to be output by the assembler. The following output types can be specified as <sub>.

Table A-12. Output Types Specifiable for show Option

Output Type	Description
conditionals	The statements for which the specified condition is not satisfied in conditional assembly are also output to a source list file.
definitions	The information before replacement specified by **.DEFINE** is output to a source list file.
expansions	The macro expansion statements are output to a source list file.

Microcontroller Options

The following microcontroller options are available.

- -cpu
- -endian
- -fint_register
- -base
- -patch
- -pic
- -pid
- -nouse_pid_register

-cpu

[Format]

```
-cpu={ rx600 | rx200 }
```

- [Default]
  The default for this option is determined based on the environment variable **RX_CPU**.

[Description]

- This option specifies the CPU type for the instruction code to be generated.
- When **cpu=rx600** is specified, a relocatable file for the RX600 Series is generated.
- When **cpu=rx200** is specified, a relocatable file for the RX200 Series is generated.

[Remarks]

- Suboptions will be added depending on the microcontroller products developed in the future.
- When **rx200** is specified, writing floating-point operation instructions (**FADD**, **FCMP**, **FDIV**, **FMUL**, **FSUB**, **FTOI**, **ITOF**, and **ROUND**) which are not supported by the RX200 Series or writing **FPSW** in control registers for the **MVTC**, **MVFC**, **PUSHC**, and **POPC** instructions will cause an error.

-endian

[Format]

```
-endian={ big  |  little }
```

- [Default]
  The default for this option is **endian=little**.

[Description]

- When **endian=big** is specified, data bytes are arranged in big endian.
  When **endian=little** is specified, data bytes are arranged in little endian.

-fint_register

[Format]

```
-fint_register = {0 | 1 | 2 | 3 | 4 }
```

- [Default]
  The default for this option is **fint_register=0**.

[Description]

- This option outputs to the relocatable file the information about the general registers that are specified to be used only for fast interrupts through the same-name option in the compiler.

[Remarks]

- Be sure to set this option to the same value for all assembly processes in the project. If a different setting is made, correct operation is not guaranteed.
- Do not use a general register dedicated to fast interrupts for other purposes in assembly-language files. If such a register is used for any other purpose, correct operation is not guaranteed.
- If a register specified by this option is also specified by the **base** option, an error will be output.

-base

[Format]

```
-base = {   rom = <register>
          | ram = <register>
          | <address> = <register>}
                        <register> = {R8 to R13}
```

[Description]

- This option outputs to the relocatable file the information about the general register that is specified to be used only as a base address register through the same-name option in the compiler.

[Remarks]

- Be sure to set this option to the same value for all assembly processes in the project. If a different setting is made, correct operation is not guaranteed.
- Do not use a general register specified by this option for other purposes than a base address register. If such a register is used for any other purpose, correct operation is not guaranteed.
- If a single general register is specified for different areas, an error will be output.
- If a general register specified by the **fint_register** option is also specified by this option, an error will be output.

-patch

[Format]

```
-patch = { rx610 }
```

[Description]

- This option is used to avoid a problem specific to the CPU type.
- When **-patch=rx610** is specified, the **MVTIPL** instruction which causes a problem in the RX610 Group is handled as an undefined instruction. The **MVTIPL** instruction will not be recognized as an instruction and the error message E0552113 will be output.

-pic

[Format]

```
-pic
```

- [Default]
  This option generates a relocatable object indicating that code was generated with the PIC function disabled.

[Description]

- This option generates a relocatable object indicating that code was generated with the PIC function enabled.

[Remarks]

- Even if code conflicting with this option is written in the assembly code, it will not be checked.
- A relocatable object with the PIC function enabled cannot be linked with a relocatable object with the PIC function disabled.
- For the PIC function, also refer to Usage of PIC/PID Function.

-pid

[Format]

```
-pid[={ 16 | 32 }]
```

- [Default]
  This option generates a relocatable object indicating that code was generated with the PID function disabled.

[Description]

- This option generates a relocatable object indicating that code was generated with the PID function enabled.
- <PID register>
  - Based on the rules in the following table, one register from among R9 to R13 is selected according to the specification of the **fint_register** option. If the **fint_register** option is not specified, R13 is selected.

Table A-13. Correspondences between fint_register Options and PID Registers

fint_register Option	PID Register
No **fint_register** specification	R13
fint_register = 0	
fint_register = 1	R12
fint_register = 2	R11
fint_register = 3	R10
fint_register = 4	R9

  - The PID register can be used only for the purpose of PID access.
- <Parameters>
  - The meaning of a parameter is the same as that for the compiler option with the same name.

[Remarks]

- Even if code conflicting with PID is written in the assembly code, it will not be checked.
- A relocatable object with the PID function enabled cannot be linked with a relocatable object with the PID function disabled.
- If a PID register specified by the **pid** option is also specified by the **base** option, error F0553111 will be output.
- If the **pid** option and **nouse_pid_register** option are selected simultaneously, error F0553103 will be output.
- For the PID function, also refer to Usage of PIC/PID Function.

-nouse_pid_register

[Format]

```
-nouse_pid_register
```

[Description]

- This option generates a relocatable object that was generated without using the PID register.
- If the PID register is used in the assembly-language source file, error message E0552058 will be output.
- A master program called by an application program in which the PID function is enabled needs to be assembled with this option. At this time, if the **fint_register** option is selected in the application program, the same parameter **fint_register** should also be selected in the master program.

[Remarks]

- If the **nouse_pid_register** option and **pid** option are selected simultaneously, error F0553103 will be output.
- If a register specified by the **nouse_pid_register** option is also specified by the **base** option, error F0553112 will be output.
- For the PID function, also refer to Usage of PIC/PID Function.

Other Options

The following other options are available.

- -logo
- -nologo
- -subcommand
- -euc
- -sjis
- -latin1
- -big5
- -gb2312

-logo

[Format]

```
-logo
```

- [Default]
  The copyright notice is output.

[Description]

- The copyright notice is output.

-nologo

[Format]

```
-nologo
```

- [Default]
  The copyright notice is output.

[Description]

- When the **nologo** option is specified, output of the copyright notice is disabled.

-subcommand

[Format]

```
-subcommand=<subcommand file name>
```

[Description]

- When the **subcommand** option is specified, the assembler options specified in a subcommand file are used at assembler startup. Specify options in a subcommand file in the same format as in the command line.

[Example]

- Contents of subcommand file **opt.sub**:

```
-listfile
-debug
```

- Command line specifications:
- When options are specified in the command line as shown (1) below, the assembler interprets them as shown in (2).

```
(1) asrx -endian=big -subcommand=opt.sub test.src
(2) asrx -endian=big -listfile -debug test.src
```

-euc

[Format]

```
-euc
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **sjis** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **euc** code.

-sjis

[Format]

```
-sjis
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **sjis** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **sjis** code.

-latin1

[Format]

```
-latin1
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **sjis** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **latin1** code.

-big5

[Format]

```
-big5
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **BIG5** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **big5** code.

-gb2312

[Format]

```
-gb2312
```

- [Default]
  This option specifies the character code to handle the characters in strings, character constants, and comments in **GB2312** code.

[Description]

- This option specifies the character code to handle the characters in strings, character constants, and comments in **gb2312** code.

(3) Optimizing Linkage Editor (rlink) Options

Classification	Option	Description
Input Options	-Input	Specifies relocatable files.
	-library	Specifies library files.
	-binary	Specifies binary files.
	-define	Specifies symbol definitions.
	-entry	Specifies an entry symbol or entry address.
	-noprelink	Selects non-initiation of the prelinker.
Output Options	-form	Selects the output file format.
	-debug	Debugging information is output to load module files.
	-sdebug	Debugging information is output to the **.dbg** file.
	-nodebug	Debugging information is not output.
	-record	Selects the record size.
	-rom	Specifies the section mapping from ROM to RAM.
	-output	Specifies the names of files to be output.
	-map	Outputs an external symbol-allocation information file.
	-space	Data are output to fill unused ranges of memory.
	-message	Information-level messages are output.
	-nomessage	The output of messages is disabled.
	-msg_unused	Messages are output to indicate the presence of externally defined symbols to which there is no reference.
	-byte_count	Specifies the number of bytes in a data record.
	-crc	Specifies the format for output of the CRC code.
	-padding	Padding data are included at the end of each section.
	-vectn	Assigns an address to the specified vector number in the variable vector table (for the RX Family and M16C Family).
	-vect	Assigns an address to an unused area in the variable vector table (for the RX Family and M16C Family).
	-jump_entries_for_pic	Outputs a jump table file (for the PIC function of the RX Family).
List Options	-list	A linkage list file is output.
	-show	Selects the contents to be output in the linkage list file.

Classification	Option	Description
Optimize Options	-optimize	Selects the items to be optimized at linkage.
	-nooptimize	Selects no optimization at linkage.
	-samesize	Specifies the minimum size for unification of the same codes.
	-symbol_forbid	Specifies symbols for which unreferenced symbol deletion is disabled.
	-samecode_forbid	Specifies symbols for which same code unification is disabled.
	-variable_forbid	Specifies symbols for which short absolute addressing mode is disabled.
	-function_forbid	Specifies symbols for which indirect addressing is disabled.
	-section_forbid	Specifies a section where optimization is disabled.
	-absolute_forbid	Specifies an address range where optimization is disabled.
Section Options	-start	Specifies a section start address.
	-fsymbol	Specifies the section where an external defined symbol will be placed in the output file.
	-aligned_section	Specifies the section alignment value as 16 bytes.
Verify Options	-cpu	Checks addresses for consistency.
	-contiguous_section	Specifies sections that will not be divided.
Other Options	-s9	Selects the output of an s9 record at the end of the file.
	-stack	Selects the output of a stack-usage information file.
	-compress	Debugging information are compressed.
	-nocompress	Debugging information are not compressed.
	-memory	Selects the amount of memory to be used in linkage.
	-rename	Specifies symbol names and section names to be modified.
	-delete	Specifies symbol names and module names to be deleted.
	-replace	Specifies library modules to be replaced.
	-extract	Specifies modules to be extracted from library files.
	-strip	Debugging information is deleted from absolute files and library files.
	-change_message	Specifies changes to the levels of messages (information, warning, and error).
	-hide	Name information on local symbols is deleted.
	-total_size	The total sizes of sections after linkage are sent to standard output.
Subcommand File Option	-subcommand	Specifies a file from which to include command options.
Options Other Than Above	-logo	Selects the output of copyright information.
	-nologo	Selects the non-output of copyright information.
	-end	Selects the execution of option strings specified before **END**.
	-exit	Specifies the end of option specification.

Input Options

The following input options are available.

- -Input
- -library
- -binary
- -define
- -entry
- -noprelink

-Input

[Format]

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

[Description]

- Specifies an input file. Two or more files can be specified by separating them with a comma (,) or space.
- Wildcards (* or ?) can also be used for the specification. String literals specified with wildcards are expanded in alphabetical order. Expansion of numerical values precedes that of alphabetical letters. Uppercase letters are expanded before lowercase letters.
- Specifiable files are object files output from the compiler or the assembler, and relocatable or absolute files output from the optimizing linkage editor. A module in a library can be specified as an input file using the format of <library name>(<module name>). The module name is specified without an extension.
- If an extension is omitted from the input file specification, **obj** is assumed when a module name is not specified and **lib** is assumed when a module name is specified.

[Examples]

```
input=a.obj lib1(e)       ; Inputs a.obj and module e in lib1.lib.
input=c*.obj              ; Inputs all .obj files beginning with c.
```

[Remarks]

- When **form=object** or **extract** is specified, this option is unavailable.
- When an input file is specified on the command line, **input** should be omitted.

-library

[Format]

```
-library = <file name>[,...]
```

[Description]

- Specifies an input library file. Two or more files can be specified by separating them with a comma (,).
- Wildcards (* or ?) can also be used for the specification. String literals specified with wildcards are expanded in the alphabetical order. Expansion of numerical values precedes that of alphabetical letters. Uppercase letters are expanded before lowercase letters.
- If an extension is omitted from the input file specification, **lib** is assumed.
- If **form=library** or **extract** is specified, the library file is input as the target library to be edited.
- Otherwise, after the linkage processing between files specified for the input files are executed, undefined symbols are searched in the library file.
- The symbol search in the library file is executed in the following order: user library files with the library option specification (in the specified order), the system library files with the library option specification (in the specified order), and then the default library (environment variable **HLNK_LIBRARY1,2,3**).

[Examples]

```
library=a.lib,b   ; Inputs a.lib and b.lib.
library=c*.lib    ; Inputs all files beginning with c with the extension .lib.
```

-binary

[Format]

```
-binary = <suboption>[,...]
          <suboption>: <file name>(<section name>
                         [:<boundary alignment>][/<section attribute>][,<symbol name>])
          <section attribute>: CODE | DATA
          <boundary alignment>: 1 | 2 | 4 | 8 | 16 | 32 (default: 1)
```

[Description]

- Specifies an input binary file. Two or more files can be specified by separating them with a comma (,).
- If an extension is omitted for the file name specification, **bin** is assumed.
- Input binary data is allocated as the specified section data. The section address is specified with the **start** option. That section cannot be omitted.
- When a symbol is specified, the file can be linked as a defined symbol. For a variable name referenced by a C/C++ program, add an underscore (_) at the head of the reference name in the program.
- The section specified with this option can have its section attribute and boundary alignment specified.
- CODE or DATA can be specified for the section attribute.
- When section attribute specification is omitted, the write, read, and execute attributes are all enabled by default.
- A boundary alignment value can be specified for the section specified by this option. A power of 2 can be specified for the boundary alignment; no other values should be specified.
- When the boundary alignment specification is omitted, 1 is used as the default.

[Examples]

```
input=a.obj
start=P,D*/200
binary=b.bin(D1bin),c.bin(D2bin:4,_datab)
form=absolute
```

- Allocates **b.bin** from 0x200 as the **D1bin** section.
- Allocates **c.bin** after **D1bin** as the **D2bin** section (with boundary alignment = 4).
- Links **c.bin** data as the defined symbol _**datab**.

[Remarks]

- When **form={object | library}** or **strip** is specified, this option is unavailable.
- If no input object file is specified, this option cannot be specified.

-define

[Format]

```
-define = <suboption>[,...]
          <suboption>: <symbol name>={<symbol name> | <numerical value>}
```

[Description]

- Defines an undefined symbol forcedly as an externally defined symbol or a numerical value.
- The numerical value is specified in the hexadecimal notation. If the specified value starts with a letter from A to F, symbols are searched first, and if no corresponding symbol is 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/C++ variable name, add an underscore (_) at the head of the definition name in the program. If the symbol name is a C++ function name (except for the **main** function), enclose the definition name with the double-quotes including parameter strings. If the parameter is **void**, specify as "<function name>( )".

[Examples]

```
define=_sym1=data  ; Defines _sym1 as the same value as the externally defined symbol data.
define=_sym2=4000  ; Defines _sym2 as 0x4000.
```

[Remarks]

- When **form={object | relocate | library}** is specified, this option is unavailable.

-entry

[Format]

```
-entry = {<symbol name> | <address>}
```

[Description]

- Specifies the execution start address with an externally defined symbol or address.
- The address is specified in hexadecimal notation. If the specified value starts with a letter from A to F, symbols are searched first, and if no corresponding symbol is found, the value is interpreted as an address. Values starting with 0 are always interpreted as addresses.
- For a C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function name (except for the **main** function), enclose the definition name with double-quotes in the program including parameter strings. If the parameter is **void**, specify as "<function name>()".

[Examples]

```
entry=_main         ; Specifies main function in C/C++ as the execution start address.
entry="init()"      ; Specifies init function in C++ as the execution start address.
entry=100           ; Specifies 0x100 as the execution start address.
```

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.
- When optimization with undefined symbol deletion (**optimize=symbol_delete**) is specified, the execution start address should be specified. If it is not specified, the specification of the optimization with undefined symbol deletion is unavailable. Optimization with undefined symbol deletion is not available when an address is specified with this option.

-noprelink

[Format]

```
-noprelink
```

- [Default]
  If this option is not specified, the prelinker is initiated.

[Description]

- Disables the prelinker initiation.
- The prelinker supports the functions to generate the C++ template instance automatically and to check types at run time. When the C++ template function and the run-time type test function are not used, specify the **noprelink** option to reduce the link time.

[Remarks]

- When **extract** or **strip** is specified, this option is unavailable.
- If **form=lib** or **form=rel** is specified while the C++ template function and run-time type test are used, do not specify **noprelink**.

Output Options

The following output options are available.

- -form
- -debug
- -sdebug
- -nodebug
- -record
- -rom
- -output
- -map
- -space
- -message
- -nomessage
- -msg_unused
- -byte_count
- -crc
- -padding
- -vectn
- -vect
- -jump_entries_for_pic

-form

[Format]

```
-form = {Absolute | Relocate | Object | Library[={S | U}]} | Hexadecimal | Stype | Binary}
```

- [Default]
  When this option is omitted, the default is **form=absolute**.

[Description]

- Specifies the output format.
- Table B-14 lists the suboptions.

Table A-14. Suboptions of form Option

Suboption	Description
absolute	Outputs an absolute file
relocate	Outputs a relocatable file
object	Outputs an object file. This is specified when a module is extracted as an object file from a library with the **extract** option.
library	Outputs a library file. When **library=s** is specified, a system library is output. When **library=u** is specified, a user library is output. Default is **library=u**.
hexadecimal	Outputs a **HEX** file. For details of the HEX format, refer to HEX File Format.
stype	Outputs an **S**-type file. For details of the **S**-type format, refer to S-Type File Format.
binary	Outputs a binary file.

[Remarks]

Table B-15 shows relations between output formats and input files or other options.

Table A-15. Relations Between Output Format and Input File or Other Options

Output Format	Specified Option	Enabled File Format	Specifiable Option[Note1]
Absolute	**strip** specified	Absolute file	input, output
	Other than above	Object file Relocatable file Binary file Library file	input, library, binary, debug/nodebug, sdebug, cpu, start, rom, entry, output, map, hide, optimize/ nooptimize, samesize, symbol_forbid, samecode_forbid, section_forbid, absolute_forbid, compress, rename, delete, define, fsymbol, stack, noprelink, memory, msg_unused, show=symbol, reference, xreference, jump_entries_for_pic, aligned_section

Output Format	Specified Option	Enabled File Format	Specifiable Option[Note1]
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, noprelink, msg_unused, 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, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, section_forbid, absolute_forbid, rename, delete, define, fsymbol, stack, noprelink, record, s9[Note 2], byte_count[Note3], memory, msg_unused, show=symbol, reference, xreference, jump_entries_for_pic, aligned_section
		Absolute file	input, output, record, s9[Note2], byte_count[Note3], show=symbol, reference, xreference
Library	**strip** specified	Library file	library, output, memory[Note4], show=symbol, section
	extract specified	Library file	library, output
	Other than above	Object file Relocatable file	input, library, output, hide, rename, delete, replace, noprelink, memory[Note4], show=symbol, section

Notes 1. **message**/**nomessage**, **change_message**, **logo**/**nologo**, **form**, **list**, and **subcommand** can always be specified.

2. **s9** can be used only when **form=stype** is specified for the output format.

3. **byte_count** can be used only when **form=hexadecimal** is specified for the output format.

4. **memory** cannot be used when **hide** is specified.

-debug

[Format]

```
-debug
```

- [Default]
  When this option is omitted, debugging information is output to the output file.

[Description]

- When **debug** is specified, debugging information is output to the output file.
- If **debug** is specified and if two or more files are specified to be output with **output**, they are interpreted as **sdebug** and debugging information is output to **<first output file name>.dbg**.

[Remarks]

- When **form={object | library | hexadecimal | stype | binary}**, **strip** or **extract** is specified, this option is unavailable.

-sdebug

[Format]

```
-sdebug
```

- [Default]
  When this option is omitted, debugging information is output to the output file.

[Description]

- When **sdebug** is specified, debugging information is output to **<output file name>.dbg** file.
- If **sdebug** and **form=relocate** are specified, **sdebug** is interpreted as **debug**.

[Remarks]

- When **form={object | library | hexadecimal | stype | binary}**, **strip** or **extract** is specified, this option is unavailable.

-nodebug

[Format]

```
-nodebug
```

- [Default]
  When this option is omitted, debugging information is output to the output file.

[Description]

- When **nodebug** is specified, debugging information is not output.

[Remarks]

- When **form={object | library | hexadecimal | stype | binary}**, **strip** or **extract** is specified, this option is unavailable.

-record

[Format]

```
-record = { H16 | H20 | H32 | S1 | S2 | S3 }
```

- [Default]
  When this option is omitted, various data records are output according to each address.

[Description]

- Outputs data with the specified data record regardless of the address range.
- If there is an address that is larger than the specified data record, the appropriate data record is selected for the address.

[Remarks]

- This option is available only when **form=hexadecimal** or **stype** is specified.

-rom

[Format]

```
-rom = <suboption>[,...]
       <suboption>:  <ROM section name>=<RAM section name>
```

[Description]

- Reserves ROM and RAM areas in the initialized data area and relocates a defined symbol in the ROM section with the specified address in the RAM section.
- Specifies a relocatable section including the initial value for the ROM section.
- Specifies a nonexistent section or relocatable section whose size is 0 for the RAM section.

[Examples]

```
rom=D=R
start=D/100,R/8000
```

- Reserves **R** section with the same size as **D** section and relocates defined symbols in **D** section with the **R** section addresses.

[Remarks]

- When **form={object | relocate | library}**or **strip** is specified, this option is unavailable.

-output

[Format]

```
-output = <suboption>[,...]
          <suboption>:  <file name>[=<output range>]
          <output range>:  {<start address>-<end address> | <section name>[:...]}
```

- [Default]
  When this option is omitted, the default is <first input file name>.<default extension>.
  The default extensions are as follows:
  form=absolute: abs, form=relocate: rel, form=object: obj, form=library: lib, form=hexadecimal: hex, form=stype: mot, form=binary: bin

[Description]

- Specifies an output file name. When **form=absolute**, **hexadecimal**, **stype**, or **binary** is specified, two or more files can be specified. An address is specified in the hexadecimal notation. If the specified data starts with a letter from A to F, sections are searched first, and if no corresponding section is found, the data is interpreted as an address. Data starting with 0 are always interpreted as addresses.

[Examples]

```
output=file1.abs=0-ffff,file2.abs=10000-1ffff
```

- Outputs the range from 0 to 0xffff to **file1.abs** and the range from 0x10000 to 0x1ffff to **file2.abs**.

```
output=file1.abs=sec1:sec2,file2.abs=sec3
```

- Outputs the **sec1** and **sec2** sections to **file1.abs** and the **sec3** section to **file2.abs**.

[Remarks]

- When a file is output in section units while the CPU type is RX Family in big endian, the section size should be a multiple of 4.

-map

[Format]

```
-map [= <file name>]
```

[Description]

- Outputs the external-symbol-allocation information file that is used by the compiler in optimizing access to external variables.
- When <file name> is not specified, the file has the name specified by the **output** option or the name of the first input file, and the extension **bls**.
- If the order of the declaration of variables in the external-symbol-allocation information file is not the same as the order of the declaration of variables found when the object was read after compilations, an error will be output.

[Remarks]

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

-space

[Format]

```
-space [= {<numerical value> | Random}]
```

[Description]

- Fills the unused areas in the output ranges with random values or a user-specified hexadecimal value.
- The following unused areas are filled with the value according to the output range specification in the **output** option:
- When section names are specified for the output range:
- The specified value is output to unused areas between the specified sections.
- When an address range is specified for the output range:
- The specified value is output to unused areas within the specified address range.
- A 1-, 2-, or 4-byte value can be specified. The hexadecimal value specified to the **space** option determines the output data size. If a 3-byte value is specified, the upper digit is extended with 0 to use it as a 4-byte value. If an odd number of digits are specified, the upper digits are extended with 0 to use it as an even number of digits.
- If the size of an unused area is not a multiple of the size of the specified value, the value is output as many times as possible, then a warning message is output.

[Remarks]

- When no suboption is specified by this option, unused areas are not filled with values.
- This option is available only when **form={binary | stype | hexadecimal}** is specified.
- When no output range is specified by the **output** option, this option is unavailable.

-message

[Format]

```
-message
```

[Description]

- When **message** is specified, information-level messages are output.
- When this option is omitted, the output of information-level messages is disabled.

-nomessage

[Format]

```
-nomessage [=<suboption>[,...]]
           <suboption>:  <error number>[-<error number>]
```

- [Default]
  When this option is omitted, the output of information-level messages is disabled.

[Description]

- When **nomessage** is specified, the output of information-level messages is disabled. If an error number is specified, the output of the error message with the specified error number is disabled. A range of error message numbers to be disabled can be specified using a hyphen (-).
- Each error number consists of a component number (05), phase (6), and a four-digit value (e.g. 0004 in the case of M0560004). If the four-digit section has leading zeroes, e.g. before the 4 in the case of M0560004, these can be omitted.
- If a warning or error level message number is specified, the message output is disabled assuming that **change_message** has changed the specified message to the information level.

[Examples]

- Messages of L0004, L0200 to L0203, and L1300 are disabled to be output.

```
nomessage=4,200-203,1300
```

-msg_unused

[Format]

```
-msg_unused
```

[Description]

- Notifies the user of the externally defined symbol which is not referenced during linkage through an output message.

[Examples]

```
rlink -msg_unused a.obj
```

[Remarks]

- When an absolute file is input, this option is invalid.
- To output a message, the **message** option must also be specified.
- The linkage editor may output a message for the function that was inline-expanded at compilation. To avoid this, add a **static** declaration for the function definition.
- In any of the following cases, references are not correctly analyzed so that information shown by output messages will be incorrect.
  - There are references to constant symbols within the same file.
  - There are branches to immediate subordinate functions when optimization is specified at compilation.

-byte_count

[Format]

```
-byte_count=<numerical value>
```

[Description]

- Specifies the maximum byte count for a data record when a file is to be created in the **Intel-Hex** format. Specify a one-byte hexadecimal value (01 to FF) for the byte count. When this option is not specified, the linkage editor assumes FF as the maximum byte count when creating an **Intel-Hex** file.

[Examples]

```
byte_count=10
```

[Remarks]

- This option is invalid when the file to be created is not an **Intel-Hex**-type (**form=hex**) file.

-crc

[Format]

```
-CRc = <suboption>
       <suboption>: <address where the result is output>=<target range>
                         [/<polynomial expression>][:<endian>]
                    <address where the result is output>: <address>
                    <target range>: <start address>-<end address>[,...]
                    <polynomial expression>: { CCITT | 16 }
                    <endian>: {BIG | LITTLE}
```

[Description]

- This option is used for cyclic redundancy checking (CRC) of values from the lowest to the highest address of each target range and outputs the calculation result to the specified address.
- <endian> can be specified only when the CPU type is RX Family. When <endian> is specified, the calculation result is output to the specified address in the specified endian. When <endian> is not specified, the result is output to the specified address in the endian used in the absolute file.
- **CRC-CCITT** or **CRC-16** is selectable as a polynomial expression (default: **CRC-CCITT**).
- Polynomial expression:

```
CRC-CCITT
    X^16+X^12+X^5+1
        In bit expression: (10001000000100001)
CRC-16
    X^16+X^15+X^2+1
        In bit expression: (11000000000000101)
```

[Example]

- rlink *.obj -form=stype -start=P1,P2/1000,P3/2000
  -crc=2FFE=1000-2FFD -output=out.mot=1000-2FFF

  - **crc** option: -crc=2FFE=1000-2FFD
    - In this example, CRC will be calculated for the range from 0x1000 to 0x2FFD and the result will be output to address 0x2FFE.
      When the **space** option has not been specified, **space=0xFF** is assumed for calculation of free areas within the target range.
  - **output** option: -output=out.mot=1000-2FFF
    - Since the **space** option has not been specified, the free areas are not output to the **out.mot** file. 0xFF is used in CRC for calculation of the free areas, but will not be filled into these areas.

Notes 1. The address where the result of CRC will be output cannot be included in the target range.

2. The address where the result of CRC will be output must be included in the output range specified with the **output** option.

- rlink *.obj -form=stype -start=P1/1000,P2/1800,P3/2000
  -space=7F -crc=2FFE=1000-17FF,2000-27FF
  -output=out.mot=1000-2FFF

- **crc** option: -crc=2FFE=1000-2FFD,2000-27FF
  - In this example, CRC will be calculated for the two ranges, 0x1000 to 0x17FF and 0x2000 to 0x27FF, and the result will be output to address 0x2FFE.
    Two or more non-contiguous address ranges can be selected as the target range for CRC.
  - **space** option: -space=7F
    - The value of the **space** option (0x7F) is used for CRC in free areas within the target range.
  - **output** option: -output=out.mot=1000-2FFF
    - Since the **space** option has been specified, the free areas are output to the **out.mot** file. 0x7F will be filled into the free areas.

Notes 1. The order that CRC is calculated for the specified address ranges is not the order that the ranges have been specified. CRC proceeds from the lowest to the highest address.

2. Even if you wish to use the **crc** and **space** options at the same time, the **space** option cannot be set as **random** or a value of 2 bytes or more. Only 1-byte values are valid.

- rlink *.obj -form=stype -start=P1,P2/1000,P3/2000
  -crc=1FFE=1000-1FFD,2000-2FFF
  -output=flmem.mot=1000-1FFF

  - **crc** option: -crc=1FFE=1000-1FFD,2000-2FFF
    - In this example, CRC will be calculated for the two ranges, 0x1000 to 0x1FFD and 0x2000 to 0x2FFF, and the result will be output to address 0x1FFE.
      When the **space** option has not been specified, **space=0xFF** is assumed for calculation of free areas within the target range.
  - **output** option: -output=flmem.mot=1000-1FFF
    - Since the **space** option has not been specified, the free areas are not output to the **flmem.mot** file. 0xFF is used in CRC for calculation of the free areas, but will not be filled into these areas.

[Remarks]

- This option is invalid when two or more absolute files have been selected.
  - This option is valid only when **form={hexadecimal | stype}**.
  - When the **space** option has not been specified and the target range includes free areas that will not be output, the linkage editor assumes in CRC that 0xFF has been set in the free areas.
  - An error occurs if the target range includes an overlay area.
- Sample Code: The sample code shown below is provided to check the result of CRC figured out by the **crc** option. The sample code program should match the result of CRC by rlink.

- When the selected polynomial expression is **CRC-CCITT**:

```
typedef unsigned char    uint8_t;
typedef unsigned short   uint16_t;
typedef unsigned ong    uint32_t;

uint16_t CRC_CCITT(uint8_t *pData, uint32_t iSize)
{
    uint32_t    ui32_i;
    uint8_t     *pui8_Data;
    uint16_t    ui16_CRC = 0xFFFFu;

    pui8_Data = (uint8_t *)pData;

    for(ui32_i = 0; ui32_i < iSize; ui32_i++)
    {
        ui16_CRC  = (uint16_t)((ui16_CRC >> 8u) |
                              ((uint16_t)((uint32_t)ui16_CRC << 8u)));
        ui16_CRC ^= pui8_Data[ui32_i];
        ui16_CRC ^= (uint16_t)((ui16_CRC & 0xFFu) >> 4u);
        ui16_CRC ^= (uint16_t)((ui16_CRC << 8u) << 4u);
        ui16_CRC ^= (uint16_t)(((ui16_CRC & 0xFFu) << 4u) << 1u);
    }
    ui16_CRC = (uint16_t)( 0x0000FFFFul &
                              ((uint32_t)~(uint32_t)ui16_CRC) );
    return ui16_CRC;
}
```

- When the selected polynomial expression is **CRC-16**:

```
#define POLYNOMIAL 0xa001 // Generated polynomial expression CRC-16

typedef unsigned char    uint8_t;
typedef unsigned short   uint16_t;
typedef unsigned long    uint32_t;

uint16_t CRC16(uint8_t *pData, uint32_t iSize)
{
    uint16_t crcdData = (uint16_t)0;
    uint32_t data = 0;
    uint32_t i,cycLoop;

    for(i=0;i<iSize;i++){
        data = (uint32_t)pData[i];
        crcdData = crcdData ^ data;
        for (cycLoop = 0; cycLoop < 8; cycLoop++) {
            if (crcdData & 1) {
                crcdData = (crcdData >> 1) ^ POLYNOMIAL;
            } else {
                crcdData = crcdData >> 1;
            }
        }
    }
    return crcdData;
}
```

-padding

[Format]

```
-padding
```

[Description]

- Fills in padding data at the end of a section so that the section size is a multiple of the boundary alignment of the section.
- The file name is <output file>.jmp.
  - From the [Link Options] tab, [Fills in padding data at the end of a section] in the [Output] category

[Examples]

```
-start=P,C/0 -padding
```

- When the boundary alignment of section **P** is 4 bytes, the size of section **P** is 0x06 bytes, the boundary alignment of section **C** is 1 byte, and the size of section **C** is 0x03 bytes, two bytes of padding data is filled in section **P** to make its size become 0x08 bytes and then linkage is performed.

```
-start=P/0,C/7 -padding
```

- When the boundary alignment of section **P** is 4 bytes, the size of section **P** is 0x06 bytes, the boundary alignment of section **C** is 1 byte, and the size of section **C** is 0x03 bytes, if two bytes of padding data is filled in section **P** to make its size become 0x08 bytes and then linkage is performed, error L2321 will be output because section **P** overlaps with section **C**.

[Remarks]

- The value of the created 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.

-vectn

[Format]

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

[Description]

- Assigns the specified address to the specified vector number in the variable vector table section.
- When this option is specified, a variable vector table section is created and the specified address is set in the table even if there is no interrupt function in the source code.
- Specify a decimal value from 0 to 255 for <vector number>.
- Specify the external name of the target function for <symbol>.
- Specify the desired hexadecimal address for <address>.
- The file name is <output file>.jmp.
  - From the [Link Options] tab, [Address setting for specified vector number] in the [Output] category

[Examples]

```
-vectn=30=_f1,31=0000F100 ;Specifies the _f1 address for vector
                          ;number 30 and 0x0f100 for vector number 31
```

[Remarks]

- This option is ignored when the user creates a variable vector table section in the source program because the variable vector table is not automatically created in this case.

-vect

[Format]

```
-vect={<symbol>|<address>}
```

[Description]

- Assigns the specified address to the vector number to which no address has been assigned in the variable vector table section.
- When this option is specified, a variable vector table section is created by the linkage editor and the specified address is set in the table even if there is no interrupt function in the source code.
- Specify the external name of the target function for <symbol>.
- Specify the desired hexadecimal address for <address>.
- The file name is <output file>.jmp.
  - From the [Link Options] tab, [Address setting for unused vector area] in the [Output] category

[Remarks]

- This option is ignored when the user creates a variable vector table section in the source program because the variable vector table is not automatically created in this case.
- When the **{<symbol>|<address>}** specification is started with 0, the whole specification is assumed as an address.

-jump_entries_for_pic

[Format]

```
-jump_entries_for_pic=<section name>[,...]
```

[Description]

- Outputs an assembly-language source for a jump table to branch to external definition symbols in the specified section.
- The file name is <output file>.jmp.
  - From the [Link Options] tab, [Outputs the jump table] in the [Output] category

[Examples]

- A jump table for branching to external definition symbols in the sections **sct2** and **sct3** is output to **test.jmp**.

```
jump_entries_for_pic=sct2,sct3
output=test.abs
```

- [Example of a file output to **test.jmp**]

```
;OPTIMIZING LINKAGE EDITOR GENERATED FILE 2009.07.19
    .glb _func01
    .glb _func02
    .SECTION P,CODE
_func01:
    MOV.L #1000H,R14
    JMP   R14
_func02:
    MOV.L #2000H,R14
    JMP   R14
    .END
```

[Remarks]

- This option is invalid when **form={object | relocate| library}** or **strip** is specified.
- The generated jump table is output to the **P** section.
- Only the program section can be specified for the type of section in the section name.

List Options

The following list options are available.

- -list
- -show

-list

[Format]

```
-list [=<file name>]
```

[Description]

- Specifies list file output and a list file name.
- If no list file name is specified, a list file with the same name as the output file (or first output file) is created, with the extension **lbp** when **form=library** or **extract** is specified, or **map** in other cases.

-show

[Format]

```
-show [=<sub>[,...]]
        <sub>:{ symbol | reference | section | xreference | total_size | vector | all}
```

[Description]

- Specifies output contents of a list.
- Table B-16 lists the suboptions.
- For details of list examples, refer to Linkage List, and Library List in the user's manual.

Table A-16. Suboptions of show Option

Output Format	Suboption Name	Description
form=library or **extract** is specified.	symbol	Outputs a symbol name list in a module (when **extract** is specified).
	reference	Not specifiable.
	section	Outputs a section list in a module (when **extract** is specified).
	xreference	Not specifiable.
	total_size	Not specifiable.
	vector	Not specifiable.
	all	Not specifiable (when **extract** is specified). Outputs a symbol name list and a section list in a module (when **form=library**).
Other than **form=library** and **extract** is not specified.	symbol	Outputs symbol address, size, type, and optimization contents.
	reference	Outputs the number of symbol references.
	section	Not specifiable.
	xreference	Outputs the cross-reference information.
	total_size	Shows the total sizes of sections allocated to the ROM and RAM areas.
	vector	Outputs vector information.
	all	If **form=rel** the linkage editor outputs the same information as when **show=symbol**,**xreference**,**total_size** is specified. If **form=rel,data_stuff** have been specified, the linkage editor outputs the same information as when **show=symbol,total_size** is specified. If **form=abs** the linkage editor outputs the same information as when **show=symbol**,**reference**,**xreference**,**total_size** is specified. If **form=hex**/**stype**/**bin** the linkage editor outputs the same information as when **show=symbol**,**reference**,**xreference**,**total_size** is specified. If **form=obj**, **all** is not specifiable.

[Remarks]

- The following table shows whether suboptions will be valid or invalid by all possible combinations of options **form**, **show**, and/or **show=all**.

		Symbol	Reference	Section	Xreference	Vector	Total_size
form=abs	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Valid	Invalid	Valid	Valid	Valid
form=lib	show	Valid	Invalid	Valid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Valid	Invalid	Invalid	Invalid
form=rel	show	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Invalid	Valid[Note]	Invalid	Valid
form=obj	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Invalid	Invalid	Invalid	Invalid	Invalid
form=hex/bin/sty	show	Valid	Valid	Invalid	Invalid	Invalid	Invalid
	show=all	Valid	Valid	Invalid	Valid	Valid[Note]	Valid[Note]

Note The option is invalid if an absolute-format file is input.

- Note the following limitations on output of the cross-reference information.
  - When an absolute-format file is input, the referrer address information is not output.
  - Information about references to constant symbols within the same file is not output.
  - When optimization is specified at compilation, information about branches to immediate subordinate functions is not output.
  - When optimization of access to external variables is specified, information about references to variables other than base symbols is not output.
  - Both **show=total_size** and **total_size** output the same information.
  - When show=reference is valid, the number of references of the variable specified by #pragma address is output as 0.

Optimize Options

The following optimize options are available.

- -optimize
- -nooptimize
- -samesize
- -symbol_forbid
- -samecode_forbid
- -variable_forbid
- -function_forbid
- -section_forbid
- -absolute_forbid

-optimize

[Format]

```
-optimize [= <suboption>[,...] ]
<suboption>: { SYmbol_delete | SAMe_code | SHort_format | Branch | SPeed  | SAFe }
```

- [Default]
  When this option is omitted, the default is **optimize**.

[Description]

- When **optimize** is specified, optimization is performed for the file specified with the **goptimize** option at compilation or assembly.
- Table B-17 shows the suboptions.

Table A-17. Suboptions of optimize Option

Suboption	Description	Program to be Optimized[Note1]	
		RXC	RXA
No parameter	Provides all optimizations	O	×
symbol_delete	Deletes variables/functions that are not referenced. Always be sure to specify **#pragma entry** at compilation or the **entry** option in the optimizing linkage editor.	O	×
same_code	Creates a subroutine for the same instruction sequence.	O	×
short_format	Replaces an instruction having a displacement or an immediate value with a smaller-size instruction when the code size of the displacement or immediate value can be reduced.	O	×
branch	Optimizes branch instruction size according to program allocation information. Even if this option is not specified, it is performed when any other optimization is executed.	O	×

Suboption	Description	Program to be Optimized[Note1]	
		RXC	RXA
speed	Executes optimizations other than those reducing object speed. This suboption is the same as the following specifications: optimize=string_unify, symbol_delete, variable_access, register, short_format, or branch	O[Note2]	×
safe	Executes optimizations other than those limited by variable or function attributes. This suboption is the same as the following specifications: optimize=string_unify, register, short_format, or branch	O[Note3]	×

Notes 1. RXC: C/C++ program for RX Family,
RXA: Assembly program for RX Family

2. **symbol_delete**, **branch**, and **short_format** are valid in optimization for which **speed** was specified.

3. **short_format** and **branch** are valid in optimization for which **safe** was specified.

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.
- When a start function with **#pragma entry** or **entry** is not specified, **optimize=symbol_delete** is invalid.

-nooptimize

[Format]

```
-nooptimize
```

- [Default]
  When this option is omitted, the default is **optimize**.

[Description]

- When **pnooptimize** is specified, optimization is not performed at linkage.

-samesize

[Format]

```
-samesize = <size>
```

- [Default]
  When this option is omitted, the default is **samesize=1E**.

[Description]

- Specifies the minimum code size for the optimization with the same-code unification (**optimize=same_code**).
  Specify a hexadecimal value from 8 to 7FFF.

[Remarks]

- When **optimize=same_code** is not specified, this option is unavailable.

-symbol_forbid

[Format]

```
-symbol_forbid = <symbol name> [,...]
```

[Description]

- Disables optimization regarding unreferenced symbol deletion. For a C/C++ variable or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function, enclose the definition name in the program with double-quotes including the parameter strings. When the parameter is **void**, specify as "<function name>( )".

[Remarks]

- If optimization is not applied at linkage, this option is ignored.

-samecode_forbid

[Format]

```
-samecode_forbid = <function name> [,...]
```

[Description]

- Disables optimization regarding same-code unification. For a C/C++ variable or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function, enclose the definition name in the program with double-quotes including the parameter strings. When the parameter is **void**, specify as "<function name>( )".

[Remarks]

- If optimization is not applied at linkage, this option is ignored.

-variable_forbid

[Format]

```
-variable_forbid = <symbol name> [,...]
```

[Description]

- Disables optimization regarding short absolute addressing mode. For a C/C++ variable or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function, enclose the definition name in the program with double-quotes including the parameter strings. When the parameter is **void**, specify as "<function name>( )".

[Remarks]

- If optimization is not applied at linkage, this option is ignored.

-function_forbid

[Format]

```
-function_forbid = <function name> [,...]
```

[Description]

- Disables optimization regarding indirect addressing mode. For a C/C++ variable or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function, enclose the definition name in the program with double-quotes including the parameter strings. When the parameter is **void**, specify as "<function name>( )".

[Remarks]

- If optimization is not applied at linkage, this option is ignored.

-section_forbid

[Format]

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

[Description]

- Disables optimization for the specified section. If an input file name or library module name is also specified, the optimization can be disabled for a specific file, not only the entire section.

[Remarks]

- If optimization is not applied at linkage, this option is ignored.
- To disable optimization for an input file with its path name, type the path with the file name when specifying **section_forbid**.

-absolute_forbid

[Format]

```
-absolute_forbid = <address> [+<size>] [,...]
```

[Description]

- Disables optimization regarding address + size specification.

[Remarks]

- If optimization is not applied at linkage, this option is ignored.

Section Options

The following section options are available.

- -start
- -fsymbol
- -aligned_section

-start

[Format]

```
-start = <sub> [,...]
         <sub>: [(] <section name> [{ : | , } <section name> [,...] ] [)] [,...] [ / <address>]
```

- [Default]
  The section is allocated at 0.

[Description]

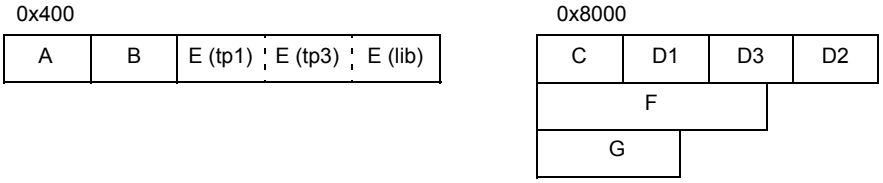
- Specifies the start address of the section. Specify an address as the hexadecimal.
- The section name can be specified with wildcards "*". Sections specified with wildcards are expanded according to the input order.
- Two or more sections can be allocated to the same address (i.e., sections are overlaid) by separating them with a colon ":".
- Sections specified at a single address are allocated in the specification order.
- Sections to be overlaid can be changed by enclosing them by parentheses "()".
- Objects in a single section are allocated in the specification order of the input file or the input library.
- If no address is specified, the section is allocated at 0.
- A section which is not specified with the **start** option is allocated after the last allocation address.

[Examples]

This example shows how sections are allocated when the objects are input in the following order (names enclosed by parentheses are sections in the objects).

tp1.obj(A,D1,E) -> tp2.obj(B,D3,F)) -> tp3.obj(C,D2,E,G) -> lib.lib(E)

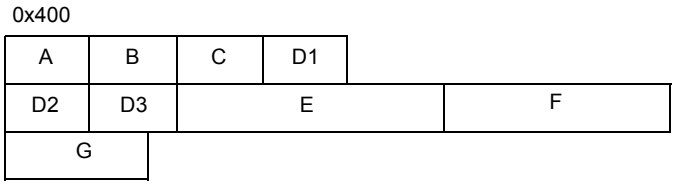
- -start=A,B,E/400,C,D*:F:G/8000



- Sections C, F, and G separated by colons are allocated to the same address.
- Sections specified with wildcards "*" (in this example, the sections whose names start with D) are allocated in the input order.
- Objects in the sections having the same name (E in this example) are allocated in the input order.

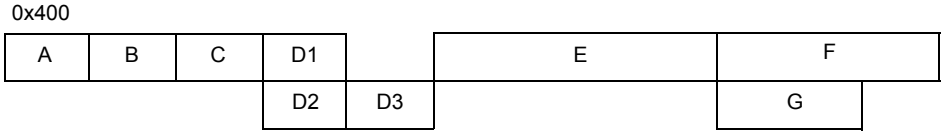
- An input library's section having the same name (E in this example) as those of input objects is allocated after the input objects.

- -start=A,B,C,D1:D2,D3,E,F:G/400



- The sections that come immediately after the colons (**A**, **D2**, and **G** in this example) are selected as the start and allocated to the same address.

- -start=A,B,C,(D1:D2,D3),E,(F:G)/400



- When the sections to be allocated to the same address are enclosed by parentheses, the sections within parentheses are allocated to the address immediately after the sections that come before the parentheses (C and E in this example).
- The section that comes after the parentheses (E in this example) is allocated after the last of the sections enclosed by the parentheses.

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.
- Parentheses cannot be nested.
- One or more colons must be written within parentheses. Parentheses cannot be written without a colon.
- Colons cannot be written outside of parentheses.
- When this option is specified with parentheses, optimization with the linkage editor is disabled.

-fsymbol

[Format]

```
-fsymbol = <section name> [,...]
```

[Description]

- Outputs externally defined symbols in the specified section to a file in the assembler directive format.
- The file name is **<output file>.fsy**.

[Examples]

- Outputs externally defined symbols in sections **sct2** and **sct3** to **test.fsy**.

```
fsymbol = sct2, sct3
output=test.abs
```

- [Output example of **test.fsy**]

```
;RENESAS OPTIMIZING LINKER GENERATED FILE 2012.07.19
;fsymbol = sct2, sct3
;SECTION NAME = sct2
 .glb_f
_f: .equ 00000000h
 .glb_g
_g: .equ 00000016h
;SECTION NAME = sct3
 .glb _main
_main: .equ 00000020h
 .end
```

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

-aligned_section

[Format]

```
-aligned_section = <section name>[,...]
```

[Description]

- Changes the alignment value for the specified section to 16 bytes.

[Remarks]

- When **form={object | relocate | library}**, **extract**, or **strip** is specified, this option is unavailable.

Verify Options

The following verify options are available.

- -cpu
- -contiguous_section

-cpu

[Format]

```
-cpu={ <memory type> = <address range> [,...] | STRIDE}
       <memory type>:  { ROm | RAm | FIX }
       <address range>:  <start address> - <end address>
```

[Description]

- When **cpu=stride** is not specified, a section larger than the specified range of addresses leads to an error.
- When **cpu=stride** is specified, a section larger than the specified range of addresses is allocated to the next area of the same memory type or the section is divided.

[Examples]

- When the **stride** suboption is not specified:

```
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF
```

  - The result is normal when **D1** and **D2** are respectively allocated within the ranges from 100 to 1FF and from 200 to 2FF. If they are not allocated within the ranges, an error will be output.

- When the **stride** suboption is specified:

```
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF,ROM=300-3FF
cpu=stride
```

  - The result is normal when **D1** and **D2** are allocated within the ROM area (regardless of whether the section is divided). A linkage error occurs when they are not allocated within the ROM area even though the section is divided.
  - Specify an address range in which a section can be allocated in hexadecimal notation. The memory type attribute is used for the inter-module optimization.
  - **FIX** for <memory type> is used to specify a memory area where the addresses are fixed (e.g. I/O area).
  - If the address range of <start>-<end> specified for **FIX** overlaps with that specified for another memory type, the setting for **FIX** is valid.
  - When <memory type> is **ROM** or **RAM** and the section size is larger than the specified memory range, sub-option **STRIDE** can be used to divide a section and allocate them to another area of the same memory type. Sections are divided in module units.

```
cpu=ROM=0-FFFF,RAM=10000-1FFFF
```

- Checks that section addresses are allocated within the range from 0 to FFFF or from 10000 to 1FFFF.
- Object movement is not provided between different attributes with the inter-module optimization.

```
cpu=ROM=100-1FF,ROM=400-4FF,RAM=500-5FF
cpu=stride
```

- When section addresses are not allocated within the range from 100 to 1FF, the linkage editor divides the sections in module units and allocates them to the range from 400 to 4FF.

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.
- When **cpu=stride** and **memory=low** are specified, this option is unavailable.
- When section **B** is divided by **cpu=stride**, the size of section **C$BSEC** increases by 8 bytes × number of divisions because this amount of information is required for initialization.

-contiguous_section

[Format]

```
-contiguous_section=<section name>[,...]
```

[Description]

- Allocates the specified section to another available area of the same memory type without dividing the section when **cpu=stride** is valid.

[Examples]

```
start=P,PA,PB/100
cpu=ROM=100-1FF,ROM=300-3FF,ROM=500-5FF
cpu=stride
contiguous_section=PA
```

- Section **P** is allocated to address 100.
- If section **PA** which is specified as **contiguous_section** is over address 1FF, section **PA** is allocated to address 300 without being divided.
- If section **PB** which is not specified as **contiguous_section** is over address 3FF, section **PB** is divided and allocated to address 500.

[Remarks]

- When **cpu=stride** is invalid, this option is unavailable.

Other Options

The following other options are available.

- -s9
- -stack
- -compress
- -nocompress
- -memory
- -rename
- -delete
- -replace
- -extract
- -strip
- -change_message
- -hide
- -total_size

-s9

[Format]

```
-s9
```

[Description]

- Outputs the **S9** record at the end even if the entry address exceeds 0x10000.

[Remarks]

- When **form=stype** is not specified, this option is unavailable.

-stack

[Format]

```
-stack
```

[Description]

- Outputs a stack consumption information file.
- The file name is **<output file name>.sni**.

[Remarks]

- When **form={object | relocate | library}** or **strip** is specified, this option is unavailable.

-compress

[Format]

```
-compress
```

- [Default]
  If this option is omitted, the debugging information is not compressed.

[Description]

- The debugging information is compressed.
- By compressing the debugging information, the debugger loading speed is improved.

[Remarks]

- When **form={object | relocate | library | hexadecimal | stype | binary}** or **strip** is specified, this option is unavailable.

-nocompress

[Format]

```
-nocompress
```

- [Default]
  If this option is omitted, the debugging information is not compressed.

[Description]

- The debugging information is not compressed.
- If the **nocompress** option is specified, the link time is reduced.

-memory

[Format]

```
-memory = [ High | Low ]
```

- [Default]
  The default for this option is **memory = high**.

[Description]

- Specifies the memory size occupied for linkage.
- When **memory = high** is specified, the processing is the same as usual.
- When **memory = low** is specified, the linkage editor loads the information necessary for linkage in smaller units to reduce the memory occupancy. This increases file accesses and processing becomes slower when the occupied memory size is less than the available memory capacity.
- **memory = low** is effective when processing is slow because a large project is linked and the memory size occupied by the linkage editor exceeds the available memory in the machine used.

[Remarks]

- When one of the following options is specified, the **memory=low** option is unavailable:
- When **form=absolute**, **hexadecimal**, **stype**, or **binary** is specified:
  compress, **delete**, **rename**, **map**, **stack**, **cpu=stride**, or
  list and **show[={reference | xreference}]** are specified in combination.
- When **form=library** is specified:
  delete, **rename**, **extract**, **hide**, or **replace**
- When **form=object** or **relocate** is specified:
  extract
- Some combinations of this option and the input or output file format are unavailable.

-rename

[Format]

```
-rename = <suboption> [,...]
          <suboption>: {[<file>] (<name> = <name> [,...])
                       | [<module>] (<name> = <name> [,...] ) }
```

[Description]

- Modifies an external symbol name or a section name.
- Symbol names or section names in a specific file or library in a module can be modified.
- For a C/C++ variable name, add an underscore (_) at the head of the definition name in the program.
- When a function name is modified, the operation is not guaranteed.
- If the specified name matches both section and symbol names, the symbol name is modified.
- If there are several files or modules of the same name, the priority depends on the input order.

[Examples]

```
rename=(_sym1=data)          ; Modifies _sym1 to data.
rename=lib1(P=P1)            ; Modifies the section P to P1
                             ; in the library module lib1.
```

[Remarks]

- When **extract** or **strip** is specified, this option is unavailable.
- When **form=absolute** is specified, the section name of the input library cannot be modified.
- Operation is not guaranteed if this option is used in combination with compile option **-merge_files**.

-delete

[Format]

```
-delete = <suboption> [,...]
          <suboption>: {[<file>] (<name>[,...]) | <module>}
```

[Description]

- Deletes an external symbol name or library module.
- Symbol names or modules in the specified file can be deleted.
- For a C/C++ variable name or C function name, add an underscore (_) at the head of the definition name in the program. For a C++ function name, enclose the definition name in the program with double-quotes including the parameter strings. If the parameter is **void**, specify as "<function name>()". If there are several files or modules of the same name, the file that is input first is applied.
- When a symbol is deleted using this option, the object is not deleted but the attribute is changed to the internal symbol.

[Examples]

```
delete=(_sym1)                ; Deletes the symbol _sym1 in all files.
delete=file1.obj(_sym2)       ; Deletes the symbol _sym2 in the file file1.obj.
```

[Remarks]

- When **extract** or **strip** is specified, this option is unavailable.
- When **form=library** has been specified, this option deletes modules.
- When **form={absolute|relocate|hexadecimal|stype|binary}** has been specified, this option deletes external symbols.
- Operation is not guaranteed if this option is used in combination with compile option **-merge_files**.

-replace

[Format]

```
-replace = <suboption> [,...]
           <suboption>: <file name> [ ( <module name> [,...] ) }
```

[Description]

- Replaces library modules.
- Replaces the specified file or library module with the module of the same name in the library specified with the **library** option.

[Examples]

```
replace=file1.obj           ; Replaces the module file1 with the module file1.obj.
replace=lib1.lib(mdl1)      ; Replaces the module mdl1 with the module mdl1
                            ; in the input library file lib1.lib.
```

[Remarks]

- When **form={object | relocate | absolute | hexadecimal | stype | binary}**, **extract**, or **strip** is specified, this option is unavailable.
- Operation is not guaranteed if this option is used in combination with compile option **-merge_files**.

-extract

[Format]

```
-extract = <module name> [,...]
```

[Description]

- Extracts library modules.
- Extracts the specified library module from the library file specified using the **library** option.

[Examples]

```
extract=file1           ; Extracts the module file1.
```

[Remarks]

- When **form={absolute | hexadecimal | stype | binary}** or **strip** is specified, this option is unavailable.
- When **form=library** has been specified, this option deletes modules.
- When **form={absolute|relocate|hexadecimal|stype|binary}** has been specified, this option deletes external symbols.

-strip

[Format]

```
-strip
```

[Description]

- Deletes debugging information in an absolute file or library file.
- When the **strip** option is specified, one input file should correspond to one output file.

[Examples]

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

- Deletes debugging information of **file1.abs**, **file2.abs**, and **file3.abs**, and outputs this information to **file1.abs**, **file2.abs**, and **file3.abs**, respectively. Files before debugging information is deleted are backed up in **file1.abk**, **file2.abk**, and **file3.abk**.

[Remarks]

- When **form={object | relocate | hexadecimal | stype | binary}** is specified, this option is unavailable.

-change_message

[Format]

```
-change_message = <suboption> [,...]
                  <suboption>: <error level> [= <error number> [-<error number>] [,...] ]
                  <error level>: {Information | Warning | Error}
```

[Description]

- Modifies the level of information, warning, and error messages.
- Specifies the execution continuation or abort at the message output.
- When a message number is specified, the error level of the message with the specified error number changes to the given level.
- A range of error message numbers can be specified by using a hyphen (-).
- Each error number must consist of a component number (05), phase (6), and a four-digit value (e.g. 2310 in the case of E0562310).
- If no error number is specified, all messages will be changed to the specified level.

[Examples]

```
change_message=warning=2310
```

- This changes E0562310 to a warning-level message so that linkage proceeds even if E0562310 is output.

```
change_message=error
```

- This changes all information and warning messages to error level messages.
  When a message is output, the execution is aborted.

-hide

[Format]

```
-hide
```

[Description]

- Deletes local symbol name information from the output file. Since all 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 symbol names 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

Note The entry function name is not hidden.

[Examples]

- The following is a C source example in which this option is valid:

```
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 function 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);
}
```

[Remarks]

- This option is available only when the output file format is specified as **absolute**, **relocate**, or **library**.
- When the input file was compiled or assembled with the **goptimize** option specified, this option is unavailable if the output file format is specified as **relocate** or **library**.
- To use this option with the external variable access optimization, do not use this option for the first linkage, and use it only for the second linkage.
- The symbol names in the debugging information are not deleted by this option.

-total_size

[Format]

```
-total_size
```

[Description]

- Sends total sizes of sections after linkage to standard output. The sections are categorized as follows, with the overall size of each being output.
- 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.

[Remarks]

- The **show=total_size** option must be used if total sizes of sections are to be output in the linkage listing.
- 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.

Subcommand File Option

The following subcommand file option is available.

- -subcommand

-subcommand

[Format]

```
-subcommand = <file name>
```

[Description]

- Specifies options with a subcommand file.
- The format of the subcommand file is as follows:
  <option> { = | ∆ } [<suboption> [,…]] [∆&] [;<comment>]
- The option and suboption are separated by an "=" sign or a space.
- For the **input** option, suboptions are separated by a space.
- One option is specified per line in the subcommand file.
- If a subcommand description exceeds one line, the description can be allowed to overflow to the next line by using an ampersand (&).
- The **subcommand** option cannot be specified in the subcommand file.

[Examples]

- Command line specification:

```
rlink file1.obj -sub=test.sub file4.obj
```

- Subcommand specification:

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

- Option contents specified with a subcommand file are expanded to the location at which the subcommand is specified on the command line and are executed.
- The order of file input is **file1.obj**, **file2.obj**, **file3.obj**, and **file4.obj**.

Options Other Than Above

The following options other than above are available.

- -logo
- -nologo
- -end
- -exit

-logo

[Format]

```
-logo
```

- \[Default]
  When this option is omitted, the copyright notice is output.

[Description]

- The copyright notice is output.

-nologo

[Format]

```
-nologo
```

- [Default]
  When this option is omitted, the copyright notice is output.

[Description]

- Output of the copyright notice is disabled.

-end

[Format]

```
-end
```

[Description]

- Executes option strings specified before **END**. After the linkage processing is terminated, option strings that are specified after **END** are input and the linkage processing is continued.
- This option cannot be specified on the command line.

[Examples]

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

- Executes the processing from (1) to (3) and outputs **a.abs**. Then executes the processing from (4) to (6) and outputs **a.mot**.

-exit

[Format]

```
-exit
```

[Description]

- Specifies the end of the option specifications.
- This option cannot be specified on the command line.

[Examples]

- Command line specification:

```
rlink -sub=test.sub -nodebug
```

- test.sub:

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

- Executes the processing from (1) to (3) and outputs **a.abs**.
- The **nodebug** option specified on the command line after **exit** is executed is ignored.

(4) Library Generator Options

Classification	Option	Description
Library Options	-head	Specifies a configuration library.
	-output	Specifies an output library file name.
	-nofloat	Creates a simple I/O function.
	-reent	Creates a reentrant library.
	-lang	Selects the set of functions available from the C standard library.
	-simple_stdio	Creates a functionally cut down version of the set of I/O functions.
	-logo -nologo	Outputs the copyright. Disables output of the copyright.

Library Options

The following library options are available.

- -head
- -output
- -nofloat
- -reent
- -lang
- -simple_stdio
- -logo
- -nologo

-head

[Format]

```
-head=<sub>[,...]
<sub>:{ all | runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios |
        new | complex | cppstring | c99_complex | fenv | inttypes | wchar | wctype}
```

- [Default]
  The default for this option is **head=all**.

[Description]

- This option specifies a configuration file with a header file name.
- When **head=all** is specified, all header file names will be configured.
- The runtime library is always configured.

-output

[Format]

```
-output=<file name>
```

- [Default]
  The default for this option is **output=stdlib.lib**.

[Description]

- This option specifies an output file name.

-nofloat

[Format]

```
-nofloat
```

[Description]

- This option creates simple I/O functions that do not support the conversion of floating-point numbers (%f, %e, %E, %g, %G).
- When inputting or outputting files that do not require the conversion of floating-point numbers, ROM can be saved.
  Target functions: fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, and vsprintf

[Remarks]

- In a library created with this option specified, correct operation cannot be guaranteed when floating-point numbers are input to or output from the target functions.

-reent

[Format]

```
-reent
```

[Description]

- This option creates reentrant libraries. Note that the rand and srand functions are not reentrant libraries.

[Remarks]

- When reentrant libraries are linked, use **#define** to define the macro name of **_REENTRANT** before including standard include files in the program or use the **define** option to define **_REENTRANT** at compilation.

-lang

[Format]

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

- [Default]
  The default for this option is **lang=c**.

[Description]

- This option selects which functions are to be usable in the C standard library.
- When **lang=c** is specified, only the functions conforming to the **C89** standard are included in the C standard library, and the extended functions of the **C99** standard are not included. When **lang=c99** is specified, the functions conforming to the **C89** standard and the functions conforming to the **C99** standard are included in the C standard library.

[Remarks]

- There are no changes in the functions included in the C++ and EC++ standard libraries.
- When **lang=c99** is specified, all functions including those specified by the **C99** standard can be used. Since the number of available functions is greater than when **lang=c** is specified, however, generating a library may take a long time.

-simple_stdio

[Format]

```
-simple_stdio
```

[Description]

- This option creates a functional cutdown version of I/O functions.
- The functional cutdown version does not include the conversion of floating-point numbers (same as the function not supported with the **nofloat** option), the conversion of **long long** type, and the conversion of 2-byte code. When inputting or outputting files that do not require these functions, ROM can be saved.
  Target functions:fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, and vsprintf

[Remarks]

- In a library created with this option specified, correct operation cannot be guaranteed when a cutdown function is used in the target functions.
- This function is disabled during C++ and EC++ program compilation.

-logo

[Format]

```
-logo
```

- [Default]
  When this option is omitted, the copyright notice is output.

[Description]

- The copyright notice is output.

-nologo

[Format]

```
-nologo
```

- [Default]
  When this option is omitted, the copyright notice is output.

[Description]

- Output of the copyright notice is disabled.

Compiler Options That Become Invalid

In addition to the options in (4) Library Generator Options, the C/C++ compiler options can be specified in the library generator as options used for library compilation. However, the options listed below are invalid; they are not selected at library compilation.

Table A-18. Invalid Options

No.	Options that Become Invalid	Conditions for Invalidation	Library Configuration When Made Invalid
1	lang	Always invalid	None
2	include	Always invalid	None
3	define	Always invalid	None
4	undefined	Always invalid	None
5	message nomessage	Always invalid	nomessage
6	change_message	Always invalid	None
7	file_inline_path	Always invalid	None
8	comment	Always invalid	None
9	check	Always invalid	None
10	output	Always invalid	output=obj
11	noline	Always invalid	None
12	debug nodebug	Always invalid	nodebug
13	listfile nolistfile show	Always invalid	nolistfile
14	file_inline	Always invalid	None
15	asmcmd	Always invalid	None
16	lnkcmd	Always invalid	None
17	asmopt	Always invalid	None
18	lnkopt	Always invalid	None
19	logo nologo	Always invalid	nologo
20	euc sjis latin1 utf8	Always invalid	None
21	outcode	Always invalid	None
22	subcommand	Always invalid	None
23	alias	Always invalid	alias=noansi

No.	Options that Become Invalid	Conditions for Invalidation	Library Configuration When Made Invalid
24	pic pid	lang=cpp or at C++ source compilation[Note1]	None
25	ip_optimize	Always invalid	None
26	merge_files	Always invalid	None
27	whole_program	Always invalid	None
28	big5 gb2312	Always invalid [Note2]	None

Notes 1. Warning W0511171 is output.

2. Error F0593305 is output. (This library cannot be generated.)

APPENDIX B INDEX

Revision Record

Rev.	Date	Description	
		Page	Summary
Rev.1.00	Jun. 01, 2013	-	First Edition issued

CC-RX V2.00.00 User's Manual:
RX Build

Publication Date: Rev.1.00 Jun. 01, 2013

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.3

CC-RX V2.00.00

Renesas Electronics Corporation

R20UT2639EJ0100