User's Manual

CC-RX V2.01.00

ユーザーズマニュアル　RX ビルド編

対象デバイス

RX ファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2013.10

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：　輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）　、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。 たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。 なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

目　　次

第 1 章 概 説

この章では，RX ファミリ向け C/C++ コンパイラが行うコンパイル処理の概要と，プログラム開発の事例を説明します。

1.1 概 要

ビルド・ツール（CC-RX）は，本製品が提供しているコンポーネントの一種であり，GUI ベースで各種情報を設定することにより，ソースファイルからロードモジュールファイル，ライブラリファイルを，目的に応じて生成することができます。

CC-RX は，以下に示す実行ファイルで構成されています。

（1）ccrx: コンパイルドライバ

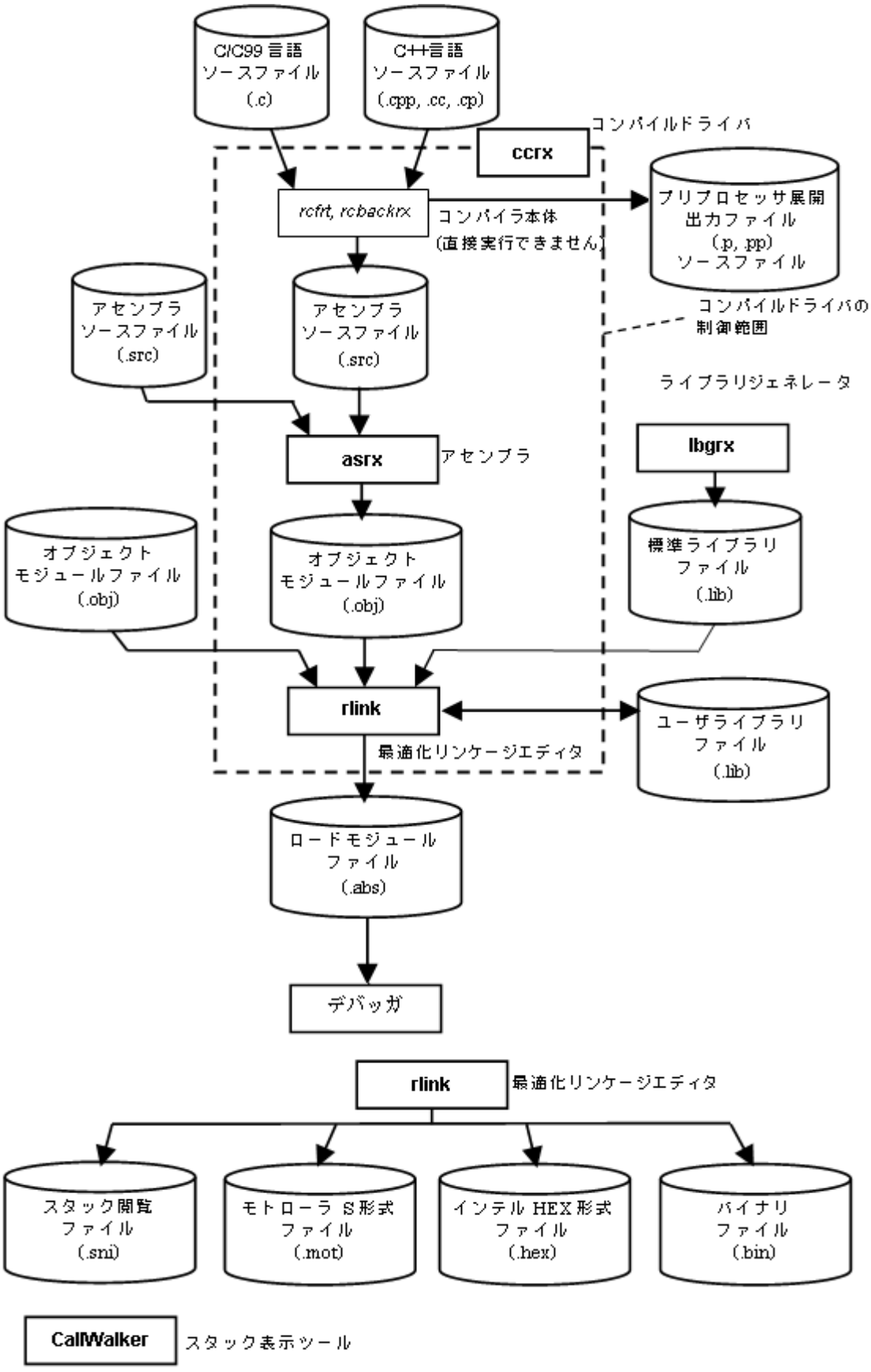
（2）asrx: アセンブラ

（3）rlink: 最適化リンケージエディタ

（4）lbgrx: ライブラリ・ジェネレータ

以下に，ビルド・ツールの処理の流れを示します。

図 1—1 ビルド・ツールの処理の流れ



1. 2 著作権について

本ソフトウェアは LLVM Release License に従い、LLVM 技術をベースに開発されました。
その他のソフトウェア構成物は、ルネサス エレクトロニクス株式会社が著作権を有します。

s

第 2 章　ビルドの出力リスト

この章では，ビルドにより各コマンドが出力する各種リストのフォーマットなどについて説明します。

2. 1　アセンブル・リスト・ファイル

アセンブラが出力するアセンブル・リスト・ファイルの内容と形式について説明します。

ソースリストファイルには，コンパイル結果およびアセンブル結果の情報を表示します。

ソースリストの構成と内容を示します。

表 2—1　ソースリストの構成と内容

No	リストファイルへ表示する情報	内容	サブオプション注	-show オプション省略時
1	ソース情報	アセンブリソースに対応して，C/C++ 言語ソースを表示	-show=source	出力しない
2	オブジェクト情報	オブジェクトプログラムの機械語，アセンブリソースコード	なし	出力する
3	統計情報	エラーの総数，ソースプログラムの行数，セクションサイズ	なし	出力する
4	コマンド指定情報	コマンドで指定されたファイル名とオプションを表示	なし	出力する

注　-listfile オプションを指定した場合に有効です。

2. 1. 1　ソース情報

ソース情報は，-show=source オプションを指定することでオブジェクト情報に埋め込まれる形で出力されます。出力例は次項を参照ください。

2. 1. 2　オブジェクト情報

オブジェクト情報の出力例を示します。

```
* 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 ***

                                 ;-ISA=RXV1

                                 ;*** 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
```

項番	説明
(1)	ロケーション情報（LOC.） アセンブル時に決定できる範囲のオブジェクトコードのロケーションアドレスを出力します
(2)	オブジェクトコード情報（OBJ.） ニーモニックに対応するオブジェクトコードを出力します

<table>
<tr><th>項番</th><th>説明</th></tr>
<tr><td>(3)</td><td>行情報（0XMDA）
アセンブラがソースを処理した結果の情報を出力します。
各記号の意味を下記に示します。
<table>
<tr><th>0</th><th>X</th><th>M</th><th>S</th><th>D</th><th>内　容</th></tr>
<tr><td>0-30</td><td></td><td></td><td></td><td></td><td>インクルードファイルのネストレベルを示します。</td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td>-show=conditions 指定時，条件アセンブルで条件が偽となった行を示します。</td></tr>
<tr><td></td><td></td><td>M</td><td></td><td></td><td>-show=expansions 指定時，マクロ命令の展開行であることを示します。</td></tr>
<tr><td></td><td></td><td>D</td><td></td><td></td><td>-show=definitions 指定時，マクロ命令の定義行であることを示します。</td></tr>
<tr><td></td><td></td><td></td><td>S</td><td></td><td>分岐距離指定子 S を指定したことを示します。</td></tr>
<tr><td></td><td></td><td></td><td>B</td><td></td><td>分岐距離指定子 B を指定したことを示します。</td></tr>
<tr><td></td><td></td><td></td><td>W</td><td></td><td>分岐距離指定子 W を指定したことを示します。</td></tr>
<tr><td></td><td></td><td></td><td>A</td><td></td><td>分岐距離指定子 A を指定したことを示します。</td></tr>
<tr><td></td><td></td><td></td><td></td><td>*</td><td>条件分岐命令に対して代替命令を選択したことを示します。</td></tr>
</table></td></tr>
<tr><td>(4)</td><td>ソース情報（SOURCE STATEMENT）
アセンブリソースファイルの内容を表示します</td></tr>
<tr><td>(5)</td><td>C/C++ ソース行番号（LineNo.）</td></tr>
<tr><td>(6)</td><td>C/C++ ソース（C-SOURCE STATEMENT）
-show=source オプションを指定した場合，C/C++ ソースを出力します</td></tr>
</table>

2. 1. 3　統計情報

統計情報の出力例を示します。

```
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
```

項番	説明
(1)	エラー，警告それぞれのメッセージ数と，ソース行の総数
(2)	セクション情報（セクション属性，サイズ，セクション名）

2. 1. 4　コンパイラのコマンド指定情報

コンパイラを起動したときのコマンドで指定されたファイル名とオプションを表示します。

コンパイラのコマンド指定情報は，リストファイルの先頭に出力されます。

コマンド指定情報の出力例を示します。

```
;*** CPU TYPE *** (1)
;-ISA=RXV1
;*** COMMAND PARAMETER *** (2)
;-output=src=C:￥tmp￥elp1894￥sample.src
;-nologo
;-show=source
;sample.c
```

項番	説明
(1)	選択されているマイコン
(2)	コンパイラに渡したファイル名とオプション]

2. 1. 5　アセンブラのコマンド指定情報

アセンブラを起動したときのコマンドで指定されたファイル名とオプションを表示します。

アセンブラのコマンド指定情報は，リストファイルの最後に出力されます。

コマンド指定情報の出力例を示します。

```
Cpu Type (1)
-ISA=RXV1
Command Parameter (2)
-output=sample.obj
-nologo
-listfile=sample.lst
```

項番	説明
(1)	アセンブラで選択されているマイコン
(2)	アセンブラに渡したファイル名とオプション

2.2 リンク・マップ・ファイル

ここでは，リンク・マップ・ファイルについて説明します。

リンク・マップとは，リンク結果の情報が書かれたもので，セクションの配置アドレスなどの情報を知ることができます。

2.2.1 リンケージリストの構成

リンケージリストの構成と内容を表 3.3 に示します。

表 2—2 リンケージリストの構成と内容

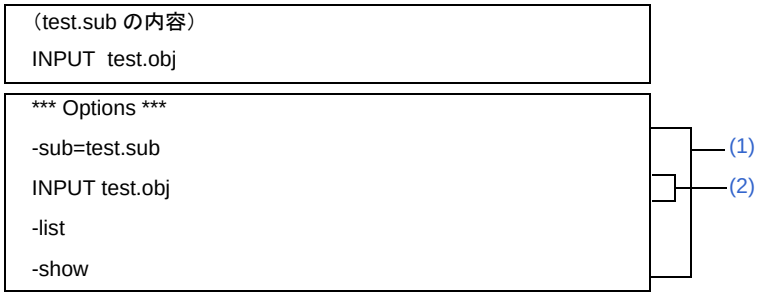
No	リストファイルへ表示する情報	内容	-show オプション[注]指定	-show オプション省略時
1	オプション情報	コマンドライン，サブコマンドで指定したオプション列を表示	なし	出力する
2	エラー情報	エラーメッセージを表示	なし	出力する
3	リンケージマップ情報	セクション名，先頭／最終アドレス，サイズ，種別を表示	なし	出力する
4	シンボル情報	静的定義シンボル名，アドレス，サイズ，種別をアドレス順に表示 -show=reference を指定した場合は，各シンボルの参照回数，最適化実行有無も表示	-show=symbol -show=reference	出力しない 出力しない
5	シンボル削除最適化情報	最適化で削除したシンボルを表示	-show=symbol	出力しない
6	クロスリファレンス情報	シンボルの参照情報を表示	-show=xreference	出力しない
7	合計セクションサイズ	RAM,ROM, およびプログラムセクションの合計サイズを表示	-show=total_size	出力しない
8	ベクタ情報	ベクタ番号とアドレスの情報を表示	-show=vector	出力しない
9	CRC 情報	CRC の演算結果および出力位置アドレスを表示	なし	CRC オプション指定時は常に出力

注 -show オプションは list オプションを指定した場合に有効です。

2.2.2 オプション情報

コマンドライン，サブコマンドファイルで指定したオプション列を出力します。

オプション情報の出力例を示します（rlink -subcommand=test.sub -list -show 指定時）。



項番	説明
(1)	コマンドライン，サブコマンドで指定したオプション列を，指定順に出力します。
(2)	サブコマンドファイル test.sub 内のサブコマンドです。

2. 2. 3 エラー情報

エラーメッセージを出力します。エラー情報の出力例を示します。

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

項番	説明
(1)	エラーメッセージを出力します

2. 2. 4 リンケージマップ情報

各セクションの先頭／最終アドレス，サイズ，種別をアドレス順に出力します。

リンケージマップ情報の出力例を示します。

```
*** 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
```

項番	説明
(1)	セクション名を表示します
(2)	先頭アドレスを表示します
(3)	最終アドレスを表示します

項番	説明
(4)	セクションサイズを表示します
(5)	セクションのアライメント数を表示します

2. 2. 5 シンボル情報

-show=symbol を指定した場合，外部定義シンボルまたは静的内部定義シンボルのアドレス，サイズ，種別をアドレス順に出力します。また，-show=reference を指定した場合は，各シンボルの参照回数，最適化実行の有無も出力します。シンボル情報の出力例を図 3.8 に示します。

```
*** Symbol List ***
SECTION= (1)
 (2)         (3)          (4)              (5)
FILE=        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
```

項番	説明
(1)	セクション名を表示します。
(2)	ファイル名を表示します
(3)	(2) のファイルに含まれる該当セクションの先頭アドレスを表示します
(4)	(2) のファイルに含まれる該当セクションの最終アドレスを表示します
(5)	(2) のファイルに含まれる該当セクションのセクションサイズを表示します
(6)	シンボル名を表示します
(7)	シンボルアドレスを表示します
(8)	シンボルサイズを表示します

項番	説明
(9)	シンボル種別を次のように表示します データ種別 　func: 関数名 　data: 変数名 　entry: エントリ関数名 　none: 未設定（ラベル，アセンブラシンボル） 宣言種別 　g: 外部定義 　l: 内部定義
(10)	シンボル参照回数を表示します。 -show=reference を指定した場合のみ表示します。 参照回数を表示しないときは，* を表示します
(11)	最適化有無を次のように表示します 　ch: 最適化によって変更されたシンボル 　cr: 最適化によって生成されたシンボル 　mv: 最適化によって移動されたシンボル

2. 2. 6　シンボル削除最適化情報

シンボル削除最適化（-optimize=symbol_delete）によって削除されたシンボルのサイズ，種別を出力します。

シンボル削除最適化情報の出力例を示します。

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

項番	説明
(1)	削除シンボル名を表示します
(2)	削除シンボル名を表示します
(3)	削除シンボルの種別を以下のように表示します データ種別 　func: 関数名 　data: 変数名 宣言種別 　g: 外部定義 　l: 内部定義

2. 2. 7　クロスリファレンス情報

-show=xreference を指定した場合，シンボルの参照情報（クロスリファレンス情報）を出力します。

クロスリファレンス情報の出力例を示します。

```
*** 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
```

項番	説明
(1)	Unit 番号。オブジェクト単位の識別番号
(2)	オブジェクト名。 リンク時の入力指定順になる
(3)	シンボル名。セクションごとに配置アドレスの昇順に出力される
(4)	シンボルの配置アドレス。 -form=relocate 指定時は，セクション先頭からの相対値となる
(5)	参照している外部シンボルのアドレスを表す。 出力形式は以下のようになる。 <Unit 番号 >(< アドレス or セクション内オフセット >:< セクション名 >)

2. 2. 8 合計セクションサイズ

ROM セクション，RAM セクション，およびプログラムセクションの合計サイズを出力します。

合計の出力例を示します。

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

項番	説明
(1)	RAM データセクションの合計サイズ
(2)	ROM データセクションの合計サイズ
(3)	プログラムセクションの合計サイズ

2. 2. 9 ベクタ情報

-show=vector を指定した場合，可変ベクタテーブルの内容を表示します。

合計の出力例を示します。

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

項番	説明
(1)	ベクタ番号
(2)	シンボルを表示します シンボルが定義されていない場合はアドレスで表示します

2. 2. 10 CRC 情報

CRC オプション指定時に CRC の演算結果および出力位置アドレスを出力します。

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

項番	説明
(1)	CRC 演算結果
(2)	CRC の演算結果の出力位置アドレス

2. 3 ライブラリ・リスト

本節では，最適化リンケージエディタが出力するライブラリリストの内容と形式について説明します。

2. 3. 1 ライブラリリストの構成

ライブラリリストの構成と内容を示します。

表 2—3 ライブラリリストの構成と内容

No	リストの作成	内容	サブオプション注	-show オプション省略時
1	オプション情報	コマンドライン，サブコマンドで指定したオプション列を表示	-	出力する
2	エラー情報	エラーメッセージを表示	-	出力する
3	ライブラリ情報	ライブラリ情報を表示	-	出力する
4	ライブラリ内モジュール，セクション，シンボル情報	ライブラリ内モジュールを表示	-	出力する
		-show=symbol を指定した場合は，モジュール内シンボル名一覧も表示	-show=symbol	出力しない
		-show=section を指定した場合は，各モジュール内セクション名，シンボル名一覧も表示	-show=section	出力しない

注 すべてのオプションは，-list オプションを指定した場合に有効です。

2. 3. 2 オプション情報

コマンドライン，サブコマンドファイルで指定したオプション列を出力します。

オプション情報の出力例を示します（rlink -subcommand=test.sub -list -show を指定した場合）。

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

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

(1)
(2)

項番	説明
(1)	コマンドライン，サブコマンドで指定したオプション列を，指定順に出力します。

項番	説明
(2)	サブコマンドファイル test.sub 内のサブコマンドです。

2. 3. 3　エラー情報

エラー，ウォーニングなどのメッセージを出力します。エラー情報の出力例を示します。

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

項番	説明
(1)	ウォーニングメッセージを出力します。

2. 3. 4　ライブラリ情報

ライブラリの種別を出力します。ライブラリ情報の出力例を示します。

```
*** Library Information ***
LIBRARY NAME=test.lib   (1)
CPU=RX600               (2)
ENDIAN=Big              (3)
ATTRIBUTE=system        (4)
NUMBER OF MODULE=1      (5)
```

項番	説明
(1)	ライブラリ名を表示します。
(2)	マイコン名を表示します。
(3)	エンディアン種別を表示します。
(4)	ライブラリファイルの属性がシステムライブラリかユーザライブラリかを表示します。
(5)	ライブラリ内モジュール数を表示します。

2. 3. 5　ライブラリ内モジュール，セクション，シンボル情報

ライブラリ内のモジュール一覧を出力します。

-show=symbol を指定した場合はモジュール内シンボル名一覧を，-show=section を指定した場合はモジュール内セクション名，シンボル名一覧を出力します。

ライブラリ内モジュール，セクション，シンボル情報の出力例を示します。

```
*** 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
```

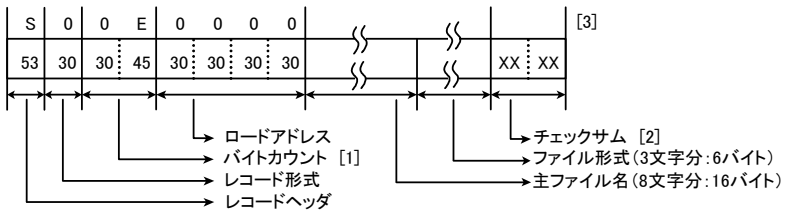
項番	説明
(1)	モジュール名を表示します。
(2)	モジュールを登録した日付を表示します。 モジュールが更新された場合は，最新の更新日付を表示します。
(3)	モジュール内セクション名を表示します。
(4)	セクション内をシンボル表示します。

2. 4 モトローラ S 形式，インテル HEX 形式ファイル

本節では，最適化リンケージエディタによって出力されるモトローラ S 形式ファイル，およびインテル HEX 形式ファイルについて説明します。

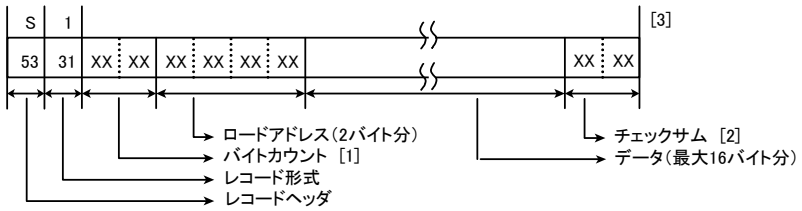
2. 4. 1 モトローラ S 形式ファイル

(a) ヘッダレコード(S0レコード)

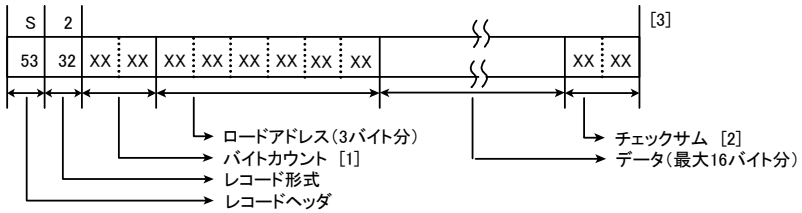


(b) データレコード(S1, S2, S3レコード)

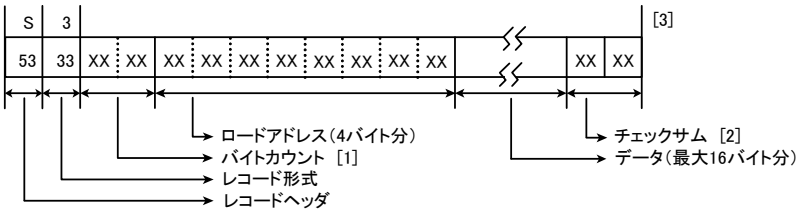
(i) ロードアドレスが0～FFFFの場合



(ii) ロードアドレスが10000～FFFFFFの場合

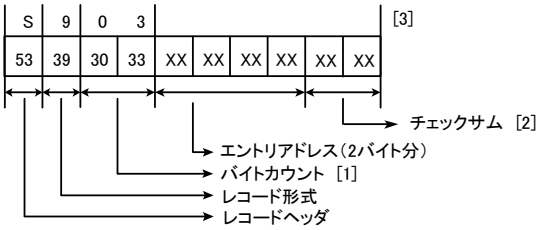


(iii) ロードアドレスが1000000～FFFFFFFFの場合

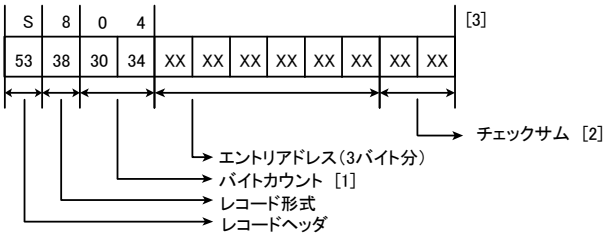


(c) エンドレコード(S9, S8, S7レコード)

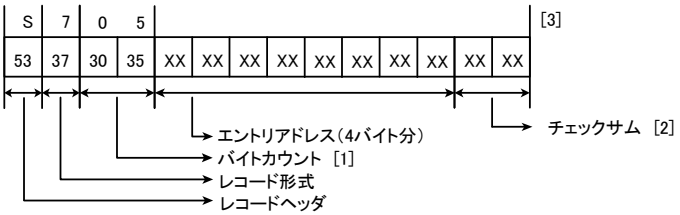
(i) エントリアドレスが0～FFFFの場合



(ii) エントリアドレスが10000～FFFFFFの場合



(iii) エントリアドレスが1000000～FFFFFFFFの場合



【注】 [1] ロードアドレス(またはエントリアドレス)からチェックサムまでのバイト数
[2] バイトカウンタからチェックサムの前までのデータ値をバイト単位に加算した結果の1の補数
[3] チェックサムの直後に改行コードが付加される

2. 4. 2 インテル HEX 形式ファイル

各データレコードの実行アドレスは以下のように求めます。

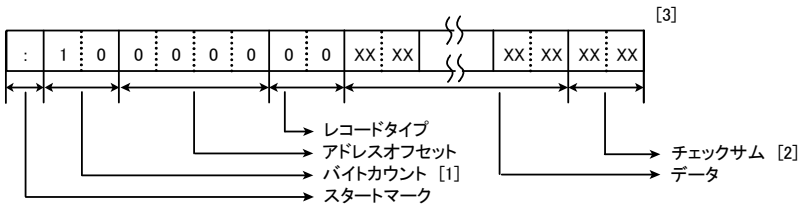
(1) セグメントアドレスの場合

(セグメントベースアドレス << 4) + (データレコードのアドレスオフセット)

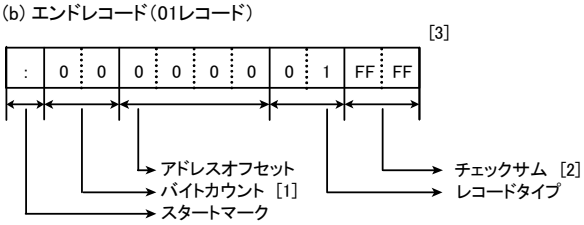
(2) リニアアドレスの場合

(リニアベースアドレス << 16) + (データレコードのアドレスオフセット)

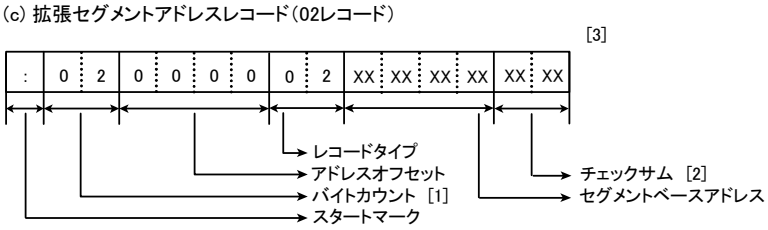
(a) データレコード(00レコード)



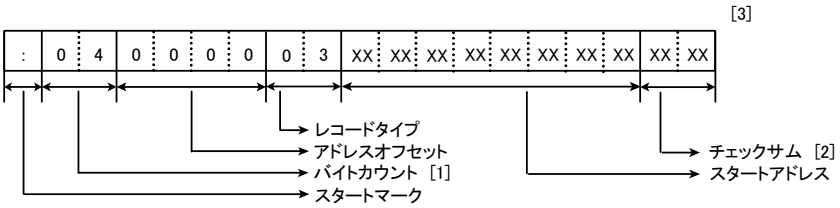
(b) エンドレコード(01レコード)



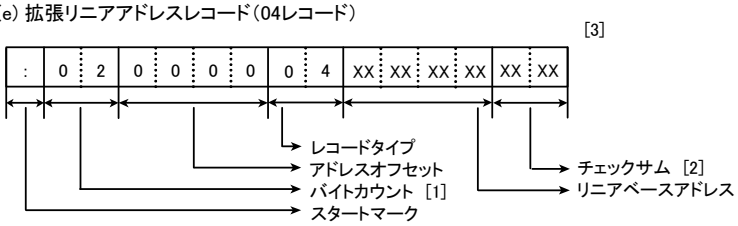
(c) 拡張セグメントアドレスレコード(02レコード)



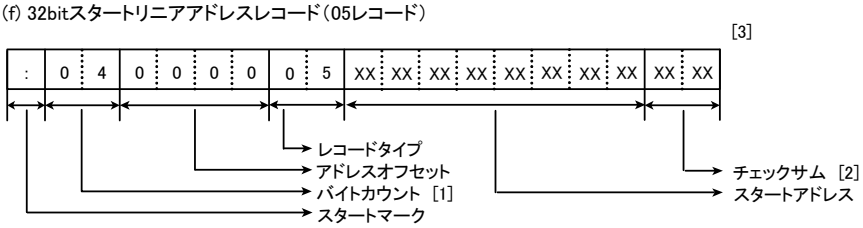
(d) スタートアドレスレコード(03レコード)



(e) 拡張リニアアドレスレコード(04レコード)



(f) 32bitスタートリニアアドレスレコード(05レコード)



【注】 [1] レコードタイプの次のデータから、チェックサムまでのバイト数
[2] バイトカウンタからチェックサムの前までのデータを、16進数で加算した結果の2の補数(下位8bitが有効)
[3] チェックサムの直後に改行コードが付加される

付録 A　コマンド・リファレンス

ここでは，ビルド・ツールに含まれる各コマンド仕様についての詳細を説明します。

A. 1　RX ファミリ用 C/C++ コンパイラ

RX ファミリ用 C/C++ コンパイラは，C 言語，C99 言語，C++ 言語やアセンブリ言語で記述したソース・プログラムから，ターゲット・システムで実行可能なファイルを生成します。

RX ファミリ用 C/C++ コンパイラでは，1 つのドライバがプリプロセッサからリンクまでのフェーズを制御します。各フェーズの処理について説明します。

(1) コンパイラ

C ソース・プログラムに対して，プリプロセス指令の処理，コメント処理，最適化を行い，アセンブラ・ソース・ファイルを生成します。

(2) プリプロセッサ

ソース・プログラム中のプリプロセス指令の処理を行います。

P オプション指定時のみ，プリプロセス処理済みファイルを出力します。

(3) 構文解析部

C ソース・プログラムの構文解析処理を行ったのち，コンパイラの内部データ表現に変換します。

(4) 最適化部

C ソース・プログラムを変換した内部データ表現に対して最適化を行います。

(5) コード生成部

内部データ表現をアセンブラ・ソース・プログラムに変換します。

(6) アセンブラ

アセンブラ・ソース・プログラムを機械語命令に変換して，再配置可能なオブジェクト・モジュール・ファイルを生成します。

(7) 最適化リンケージエディタ

オブジェクト・モジュール・ファイル，リンク・ディレクティブ・ファイル，ライブラリ・ファイルをリンクし，ターゲット・システムで実行可能なオブジェクト・ファイル（ロード・モジュール・ファイル）を生成します。

A. 1. 1 入出力ファイル

RX ファミリ用 C/C++ コンパイラの入出力ファイルを以下に示します。

表 A—1 RX ファミリ用 C/C++ コンパイラ用入出力ファイル

ファイル種別	拡張子	入出力	説明
C ソースプログラムファイル	.c	入力	C 言語，C99 言語で記述したソース・ファイル ユーザ作成ファイルです。
C++ ソースプログラムファイル	.cpp .cc .cp	入力	C++ 言語で記述したソース・ファイル ユーザ作成ファイルです。
インクルードファイル	任意	入力	ソース・ファイルで参照するファイル C 言語，C99 言語，C++ 言語，もしくはアセンブリ言語で記述したファイルです。 ユーザ作成ファイルです。
C プログラム用プリプロセッサ展開ファイル	.p	出力	入力 C 言語または C99 言語ソース・プログラムに対してプリプロセス処理を実行した結果を出力したファイル -output=prep オプション指定時に出力します。
C++ プログラム用プリプロセッサ展開ファイル	.pp	出力	入力 C++ 言語ソース・プログラムに対してプリプロセス処理を実行した結果を出力したファイル ASCII イメージファイル -output=prep オプション指定時に出力します。
アセンブリソースプログラムファイル	.src	出力	コンパイルにより，C,C99 または C++ ソースから生成したアセンブリ言語ファイル
	.src	入力	アセンブリ言語で記述したソース・ファイル
アセンブリプログラム用リストファイル	.lst	出力	アセンブル結果の情報を持つリスト・ファイル -listfile オプション指定時に出力します。 -show オプションで出力内容を選択します。
リロケータブルオブジェクトプログラムファイル	.obj	出力	機械語情報と機械語の配置アドレスに関する再配置情報，およびシンボル情報を含んだ ELF 形式ファイル
		入力	
アブソリュートロードモジュールファイル	.abs	出力	リンク結果のオブジェクト・コードの ELF 形式ファイル ヘキサ・ファイルを出力する際の入力ファイルとなります。
リンケージリストファイル	.map	出力	リンク結果の情報を持つリスト・ファイル -list オプション指定時に出力します。 -show オプションで出力内容を選択します。
ライブラリファイル	.lib	出力	複数のオブジェクト・モジュール・ファイルが登録されたファイル
		入力	
ライブラリリストファイル	.lbp	出力	ライブラリ作成結果の情報を持つリスト・ファイル -list オプション指定時に出力します。 -show オプションで出力内容を選択します。
ライブラリバックアップファイル	.lbk	出力	ライブラリジェネレータが既に存在するライブラリファイルに上書きする場合に、上書き前の内容を保存しておくファイルです。

ファイル種別	拡張子	入出力	説明
ヘキサ・ファイル（モトローラ S フォーマット）	.mot	出力	ロード・モジュール・ファイルをモトローラ S フォーマットに変換したファイル
ヘキサ・ファイル（インテル（拡張）HEX フォーマット）	.hex	出力	ロード・モジュール・ファイルをインテル（拡張）HEX フォーマットに変換したファイル
ヘキサ・ファイル（バイナリフォーマット）	.bin	出力	ロード・モジュール・ファイルをバイナリフォーマットに変換したファイル
スタック情報ファイル	.sni	出力	スタック情報ファイル -stack オプション指定時に出力します。
デバッグ情報ファイル	.dbg	出力	デバッグ情報ファイル -sdebug オプション指定時に出力します。
拡張子 td のファイルで指定された定義を含むオブジェクトファイル	.rti	出力	拡張子 td のファイルで指定された定義を含むオブジェクトファイル
呼び出し情報ファイル	.cal	出力	呼び出し情報ファイル CallWalker で出力します。
外部シンボル割り付け情報ファイル	.bls	出力	外部シンボル割り付け情報ファイル リンク時 -map オプション指定時に出力します。
	.bls	入力	外部シンボル割り付け情報ファイル コンパイル時 -map オプションの入力ファイルとして指定します。
ジャンプテーブルファイル（アセンブリ言語）	.jmp	出力	外部定義シンボルへ分岐するジャンプテーブルのアセンブラソースファイル -jump_entries_for_pic オプション指定時に出力します。
シンボルアドレスファイル（アセンブリ言語）	.fsy	出力	外部定義シンボルをアセンブラ制御命令で記述したアセンブラソースファイル -fsymbol オプション指定時に出力します。
C++ 言語機能サポートファイル	.td,.ti,.pi,.ii	出力	C++ 言語機能をサポートするための情報ファイルです。

A. 1. 2 操作方法

ここでは，RX ファミリ用 C/C++ コンパイラの操作方法について説明します。

（1）各ツールの操作方法

（a）コンパイルドライバ（ccrx）

ccrx はコンパイルドライバの起動コマンドです。

本コマンド起動により，コンパイル，アセンブル，リンクを行うことができます。

入力ファイルの拡張子が「.s」「.src」「.S」「.SRC」のいずれかである場合，コンパイルドライバはそのファイルをアセンブリ言語ファイルと解釈して，アセンブラを起動します。

入力ファイルの拡張子が「.c」「.C」のいずれかである場合，コンパイルドライバはそのファイルを C 言語ソースファイルと解釈して，コンパイラを起動します。

入力ファイルの拡張子が「.cpp」「.CPP」「.cc」「.CC」「.cp」「.CP」のいずれかである場合，コンパイルドライバはそのファイルを C++ 言語ソースファイルと解釈して，コンパイラを起動します。

これら以外の拡張子のファイルは，C 言語ソースファイルとしてコンパイルします。

【コマンド記述形式】

```
ccrx [ △ < オプション > … ][ △ < ファイル名 >[ △ < オプション > …] …]
        < オプション > : -< オプション >[=< サブオプション >[=< サブオプション >]][,…]
```

（b）アセンブラ（asrx）

asrx は，アセンブラの起動コマンドです。

【コマンド記述形式】

```
asrx [ △ < オプション > … ][ △ < ファイル名 >[ △ < オプション > …] …]
        < オプション > : -< オプション >[=< サブオプション >][,…]
```

（c）最適化リンケージエディタ（rlink）

rlink は，最適化リンケージエディタの起動コマンドです。

リンク処理だけではなく，以下に挙げる機能も含んでいます。

- リロケータブルファイル結合時の最適化
- ライブラリファイルの作成や編集
- モトローラ S 形式ファイル，インテル HEX 形式ファイル，およびバイナリファイルへのコンバート

【コマンド記述形式】

```
rlink [ △ < オプション > … ][ △ < ファイル名 >[ △ < オプション > …] …]
          < オプション > : -< オプション >[=< サブオプション >][,…]
```

（d）ライブラリジェネレータ（lbgrx）

ライブラリ・ジェネレータは、標準ライブラリを生成するツールです。標準ライブラリを生成する際、任意のコンパイル・オプションを指定することができます。

lbgrx は，ライブラリジェネレータの起動コマンドです。

【コマンド記述形式】

```
lbgrx [ △ < オプション > … ]
          < オプション > : -< オプション >[=< サブオプション >][,…]
```

（2）操作方法例

（a）コンパイル，アセンブル，リンクを 1 コマンドで実施する場合

以下の手順すべてを 1 コマンドで実施します。

- C/C++ 言語ソースファイル（tp1.c と tp2.c）を ccrx でコンパイルする
- コンパイル後，asrx でアセンブルする

- アセンブル後，rlink でリンクして，アブソリュートファイル（tp.abs）を作成する

【コマンド記述】

```
ccrx -isa=rxv1 -output=abs=tp.abs tp1.c tp2.c
```

備考 1. output オプションの出力形式指定を "-output=sty" に変えると，リンク後のファイルをモトローラ S 形式ファイルとして生成します。

2. アブソリュートファイル生成過程で生じる中間ファイル（アセンブリ言語ファイルや，リロケータブルファイル）は残りません。生成されるファイルは，output オプションで指示した形式のファイルのみです。

3. ccrx に対して，アセンブラ，最適化リンケージエディタにのみ有効なアセンブルオプションやリンクオプションを指示したい場合には，-asmcmd オプション，-lnkcmd オプション，-asmopt オプション，-lnkopt オプションを使用して指示してください。

4. リンク対象のオブジェクトは，0 番地から配置します。セクションの並び順は保証されません。配置アドレスやセクションの配置順序を指示したい場合には，-lnkcmd オプション，-lnkopt オプションを使用して最適化リンケージエディタへオプション指示してください。

（b）コンパイルとアセンブルを 1 コマンドで実施する場合

以下の手順を 1 コマンドで実施し，別コマンドでリンカを起動して，tp.abs を作成します。

- C/C++ 言語ソースファイル（tp1.c と tp2.c）を ccrx でコンパイルする
- コンパイル後，asrx でアセンブルして，リロケータブルファイル（tp1.obj, tp2.obj）を作成する

【コマンド記述】

```
ccrx -isa=rxv1 -output=obj tp1.c tp2.c
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

備考 1. ccrx に対して "-output=obj" オプションを指示すると，ccrx はリロケータブルファイルを生成します。

2. リロケータブルファイル名を変更する場合は，ccrx へ C/C++ 言語ソースファイルをひとつずつ入力する必要があります。

3. rlink の form オプションを，"-form=sty" に変えると，リンク後のファイルをモトローラ S 形式ファイルとして生成します。

（c）コンパイル，アセンブル，リンクを各々別コマンドで実施する場合

以下の個々の手順を，それぞれ 1 コマンドで実施します。

- C/C++ 言語ソースファイル（tp1.c と tp2.c）を ccrx でコンパイルして，アセンブリ言語ファイル（tp1.src, tp2.src）を作成する
- アセンブリ言語ファイル（tp1.src, tp2.src）を asrx でアセンブルして，リロケータブルファイル（tp1.obj, tp2.obj）を生成する
- リロケータブルファイル（tp1.obj, tp2.obj）を rlink でリンクして，アブソリュートファイル（tp.abs）を作成する

【コマンド記述】

```
ccrx -isa=rxv1 -output=src tp1.c tp2.c
asrx tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

備考　ccrx に対して "-output=src" オプションを指示すると，ccrx はアセンブリ言語ファイルを生成します。

（d）アセンブルとリンクを 1 コマンドで実施する場合

以下の手順すべてを 1 コマンドで実施します。

- アセンブリ言語ファイル（tp1.src, tp2.src）を asrx でアセンブルする
- アセンブル後，rlink でリンクして，アブソリュートファイル（tp.abs）を作成する

【コマンド記述】

```
ccrx -isa=rxv1 -output=abs=tp.abs tp1.src tp2.src
```

備考　リンク対象のオブジェクトは，0 番地から配置します。セクションの並び順は保証されません。配置アドレスやセクションの配置順序を指示したい場合には，-lnkcmd オプション，-lnkopt オプションを使用して最適化リンケージエディタへオプション指示してください。

（e）アセンブルとリンクを別コマンドで実施する場合

以下の個々の手順を，それぞれ 1 コマンドで実施します。

- アセンブリ言語ファイル（tp1.src, tp2.src）を asrx でアセンブルして，リロケータブルファイル（tp1.obj, tp2.obj）を生成する
- リロケータブルファイル（tp1.obj, tp2.obj）を rlink でリンクして，アブソリュートファイル（tp.abs）を作成する

【コマンド記述 1】

```
ccrx -isa=rxv1 -output=obj tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

【コマンド記述 2】

```
asrx -isa=rxv1 tp1.src tp2.src
rlink -form=abs -output=tp.abs -subcommand=cmd.sub tp1.obj tp2.obj
```

（3）環境変数 (コマンドプロンプト)

環境変数の一覧を以下に示します。

表 A―2　環境変数

No.	環境変数	説明	設定省略時の解釈
1	path	実行ファイルの格納ディレクトリを指定します。	省略不可
2	BIN_RX	ccrx を格納したディレクトリを指定します。	＜ccrx コマンドの格納ディレクトリ＞ lbgrx コマンド利用時は、省略不可

No.	環境変数	説明	設定省略時の解釈
3	ISA_RX *1	命令セットアーキテクチャを選択します。 < 命令セットアーキテクチャ > RXV1 RXV2	省略時、値は設定されません。
4	INC_RX	コンパイラのインクルードファイル格納ディレクトリを指定します。	＜ ccrx コマンドの格納ディレクトリ＞ \..\include
5	INC_RXA	アセンブラのインクルードファイル格納ディレクトリを指定します。	省略時、値は設定されません。
6	TMP_RX	テンポラリファイルを作成するディレクトリを指定します。	ccrx コマンド利用時は、%TEMP%
7	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	最適化リンケージエディタが使用するデフォルトライブラリ名を指定します。library オプションで指定したライブラリを優先してリンクします。その後未解決のシンボルがある場合、1,2,3 の順にデフォルトライブラリを検索します。	省略時、値は設定されません。
8	HLINK_TMP	最適化リンケージエディタがテンポラリファイルを作成するフォルダを指定します。この環境変数の指定がない場合は、カレントフォルダにテンポラリファイルを作成します。	省略時、値は設定されません。
9	HLINK_DIR	最適化リンケージエディタの入力ファイル格納フォルダを指定します。 input オプション、library オプションで指定したファイルの検索順序は、カレントフォルダ、HLNK_DIR 指定フォルダになります。ただし、ワイルドカードで指定したファイルは、カレントフォルダ内だけ検索します。	省略時、値は設定されません。
10	CPU_RX *1	CPU 種別を指定します。 <CPU 種別 > RX600 RX200	省略時、値は設定されません。

*1) 環境変数 ISA_RX と CPU_RX の両方を定義している場合は、ISA_RX の内容を優先します。

A. 1. 3　オプション

ここでは，RX ファミリ用 C/C++ コンパイラのオプションについて各フェーズごとに説明します。

コンパイル・フェーズ →「(1) コンパイル・オプション」参照

アセンブル・フェーズ →「(2) アセンブル・オプション」参照

リンク・フェーズ →「(3) 最適化リンケージエディタ（rlink）・オプション」参照

ライブラリ生成・フェーズ→「(4) ライブラリジェネレータ・オプション」参照

(1) コンパイル・オプション

コンパイル・フェーズのオプションの分類と説明を以下に示します。

分類	オプション	説明
ソースオプション	-lang	ソース・ファイルをコンパイルする言語を選択します。
	-include	インクルード・ファイルの取り込み先フォルダを指定します。
	-preinclude	コンパイル単位の先頭にインクルードするファイルを指定します。
	-define	マクロ定義を指定します。
	-undefine	無効化するプリデファインド・マクロを指定します。
	-message	インフォメーションレベル・メッセージを出力します。
	-nomessage	抑止するインフォメーションレベル・メッセージ番号を指定します。
	-change_message	コンパイラ出力メッセージレベル変更します。
	-file_inline_path	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】ファイル間インライン展開ファイル取り込み先フォルダの指定
	-comment	コメント（/* */）のネストを許すかどうかを選択します。
	-check	M16C(R8C),H8(H8S,H8SX),SuperH ファミリ用の既存プログラムとの互換性をチェックします。
	-misra2004	MISRA-C:2004 ルールによるソースチェックをします。
	-ignore_files_misra	MISRA-C:2004 チェック対象外のファイルを指定します。
	-check_language_extension	拡張機能の使用によって部分抑止している MISRA-C:2004 ルールのチェックを有効にします。

分類	オプション	説明
オブジェクトオプション	-output	出力ファイル形式を選択します。
	-noline	プリプロセッサ展開時に #line の出力しません。
	-debug	オブジェクト・ファイルにデバッグ情報を出力します。
	-nodebug	オブジェクト・ファイルにデバッグ情報を出力しません。
	-section	変更するセクション名を指定します。
	-stuff	変数のアライメントに応じたセクションに配置します。
	-nostuff	変数のアライメントに応じたセクションに配置しません。
	-instalign4	分岐先を 4 バイトで命令実行向け整合します。
	-instalign8	分岐先を 8 バイトで命令実行向け整合します。
	-noinstalign	分岐先を命令実行向け整合しません。
	-nouse_div_inst	除算、剰余算に DIV,DIVU,FDIV 命令を使用しません。
リストオプション	-listfile	ソース・リスト・ファイルを出力します。
	-nolistfile	ソース・リスト・ファイルを出力しません。
	-show	ソース・リスト・ファイルの内容を指定します。
最適化オプション (1/2)	-optimize	最適化レベルを選択します。
	-goptimize	モジュール間最適化用付加情報を出力します。
	-speed	実行性能重視の最適化を実施します。
	-size	コード・サイズ重視の最適化を実施します。
	-loop	ループ展開の最大展開数を指定します。
	-inline	自動インライン展開行います。
	-noinline	自動インライン展開を行いません。
	-file_inline	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】ファイル間インライン展開対象ファイルの指定
	-case	switch 文のコード展開方式を選択します。
	-volatile	外部変数を volatile 化します。
	-novolatile	外部変数を volatile 化しません。
	-const_copy	const 修飾された外部変数の定数伝播を実施します。
	-noconst_copy	const 宣言された外部変数の定数伝播を実施しません。
	-const_div	整数型定数による除算および剰余算を変換します。
	-noconst_div	整数型定数による除算および剰余算を変換しません。
	-library	ライブラリ関数の実行方法を選択します。
	-scope	最適化範囲を複数に分割してコンパイルします。
	-noscope	最適化範囲を複数に分割しないでコンパイルします。
	-schedule	パイプライン処理を考慮した命令並べ替えを行います。
	-noschedule	命令並べ替えを行いません。

分類	オプション	説明
最適化オプション (2/2)	-map	外部変数アクセス最適化を行います。
	-smap	コンパイル単位内で定義された外部変数に対し、外部変数アクセス最適化を行います。
	-nomap	外部変数アクセス最適化を行いません。
	-approxdiv	浮動小数点定数除算の乗算化を行います。
	-enable_register	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】register 指定変数を優先的にレジスタ割り付け
	-simple_float_conv	浮動小数点型、整数型間の型変換を一部省略します。
	-fpu	浮動小数点演算命令を使用します。
	-nofpu	浮動小数点演算命令を使用しません。
	-alias	ポインタ指示先の型を考慮した最適化を実施するかどうかを選択します。
	-float_order	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】浮動小数点式の演算順序変更の最適化を行います。
	-ip_optimize	大域最適化を実施します。
	-merge_files	複数ソースのコンパイル結果をひとつのオブジェクトに出力します。
	-whole_program	指定ソースファイルをプログラム全体と仮定して最適化を実施します。

分類	オプション	説明
マイコンオプション	-isa	命令セット・アーキテクチャを選択します。
	-cpu	マイコン種別を選択します。
	-endian	エンディアン選択します。
	-round	浮動小数点定数演算の丸め方式を選択します。
	-denormalize	浮動小数点定数での非正規化数の扱いを選択します。
	-dbl_size	double 型、および long double 型の精度を選択します。
	-int_to_short	int を short に、unsigned int を unsigned short に置換します。
	-signed_char	char 型を signed char 型として扱います。
	-unsigned_char	char 型を unsigned char 型として扱います。
	-signed_bitfield	ビットフィールドの符号を signed で解釈します。
	-unsigned_bitfield	ビットフィールドの符号を unsigned で解釈します。
	-auto_enum	列挙型データのサイズを自動選択するかどうかを選択します。
	-bit_order	ビットフィールドメンバの並び順を選択します。
	-pack	構造体メンバ、クラスメンバのアライメント数を 1 にします。
	-unpack	構造体メンバ、クラスメンバのアライメント数をデータのアライメントに従います。
	-exception	例外処理機能を有効にします。
	-noexception	例外処理機能を無効にします。
	-rtti	C++ 実行時型情報（dynamic_cast、typeid）を有効または、無効を選択します。
	-fint_register	高速割り込み関数でのみ使用する汎用レジスタを選択します。
	-branch	分岐幅のサイズを選択します。
	-base	ROM,RAM 用ベースレジスタ選択します。
	-patch	CPU タイプ特有の問題を回避するかどうかを選択します。
	-pic	PIC 機能を有効にします。
	-pid	PID 機能を有効にします。
	-nouse_pid_register	PID レジスタをコード生成に使用しません。
	-save_acc	割り込み関数でアキュムレータ を退避・回復します。
アセンブル・リンクオプション	-asmcmd	asrx のオプションのサブコマンドファイルを指定します。
	-lnkcmd	rlink のオプションのサブコマンドファイルを指定します。
	-asmopt	asrx のオプションを指定します。
	-lnkopt	rlink のオプションを指定します。

分類	オプション	説明
その他オプション	-logo	コピーライトを出力します。
	-nologo	コピーライトの出力しません。
	-euc	入力プログラムの文字コードを EUC コードと解釈します。
	-sjis	入力プログラムの文字コードを SJIS コードと解釈します。
	-latin1	入力プログラムの文字コードを ISO-Latin1 コードと解釈します。
	-utf8	入力プログラムの文字コードを UTF-8 コードと解釈します。
	-big5	入力プログラムの文字コードを BIG5 コードと解釈します。
	-gb2312	入力プログラムの文字コードを GB2312 コードと解釈します。
	-outcode	出力アセンブリ言語ファイルの文字コードを選択します。
	-subcommand	コマンドオプションを取り込むファイルを指定します。

ソースオプション

＜コンパイル・オプション / ソースオプション＞

ソースオプションには，次のものがあります。

- -lang
- -include
- -preinclude
- -define
- -undefine
- -message
- -nomessage
- -change_message
- -file_inline_path（※）無効オプション
- -comment
- -check
- -misra2004
- -ignore_files_misra
- -check_language_extension

-lang

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

- 省略時解釈

  拡張子が cpp，cc，cp のときには C++ 言語ソースファイルとしてコンパイルし，それ以外の場合は C（C89）言語ソースファイルとしてコンパイルします。

  ただし，拡張子が src，s の場合は本オプション指定に関わらずアセンブリ言語ファイルとして扱います。

［詳細説明］

- ソースファイルの言語を指定します。
- lang=c オプション指定時は，C （C89）言語ソースファイルとしてコンパイルします。
- lang=cpp オプション指定時は，C++ 言語ソースファイルとしてコンパイルします。
- lang=ecpp オプション指定時は，Embedded C++ 言語ソースファイルとしてコンパイルします。
- lang=c99 オプション指定時は，C （C99）言語ソースファイルとしてコンパイルします。

［備考］

- Embedded C++ 言語仕様では，catch，const_cast，dynamic_cast，explicit，mutable，namespace，reinterpret_cast，static_cast，template，throw，try，typeid，typename，using，多重継承，仮想基底クラスをサポートしていません。これらを記述した場合，エラーメッセージを出力します。
- EC++ ライブラリを使用する場合は，必ず lang=ecpp オプションを指定してください。

-include

<コンパイル・オプション / ソースオプション>

［指定形式］

```
-include = <パス名>[,…]
```

［詳細説明］

- インクルードファイルの存在するパス名を指定します。
- パス名が複数ある場合にはカンマ（,）で区切って指定することができます。
- システムインクルードファイルの検索は，include オプション指定フォルダ，環境変数 INC_RX 指定フォルダ，環境変数 BIN_RX 指定フォルダの順序で行います。
- ユーザインクルードファイルの検索は，コンパイル対象ソースファイルのあるフォルダ，include オプション指定フォルダ，環境変数 INC_RX 指定フォルダ，環境変数 BIN_RX 指定フォルダの順序で行います。

-preinclude

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-preinclude = < ファイル名 >[,…]
```

［詳細説明］

- 指定したファイルの内容をコンパイル単位の先頭に取り込みます。
- ファイル名が複数ある場合にはカンマ（,）で区切って指定することができます。
- preinclude オプション指定フォルダが複数ある場合，左に指定したものから順に検索を行います。

［備考］

- 本オプションを複数回指定した場合，指定したすべてのファイルが取り込み対象となります。

-define

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-define = <sub>[,…]
          <sub> : < マクロ名 > [= < 文字列 >]
```

［詳細説明］

- ソースファイル内で記述する #define と同等の効果を得ます。
- < マクロ名 >=< 文字列 > と記述することで < 文字列 > をマクロ名として定義できます。
- サブオプションに < マクロ名 > を単独で指定した場合は，そのマクロ名が定義されたものと仮定します。
- < 文字列 > には，名前または整定数を記述することができます。

［備考］

- 本オプションで指定したマクロ名がソース中で #define により既に定義されている場合，#define を優先します。
- 本オプションを複数回指定した場合，指定したすべてのマクロ名が有効となります。

-undefine

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-undefine = <sub>[,…]
        <sub> : < マクロ名 >
```

［詳細説明］

- ＜マクロ名＞のプリデファインドマクロを無効化します。
- マクロ名が複数ある場合にはカンマ（,）で区切って指定することができます。

［備考］

- 指定可能なプリデファインドマクロについては，RX コーディング編の「マクロ名」の項目を参照ください。
- 本オプションを複数回指定した場合，指定したすべてのマクロ名が未定義となります。

-message

<コンパイル・オプション / ソースオプション>

[指定形式]

```
-message
```

[詳細説明]

- インフォメーションレベル・メッセージを出力します。

[備考]

- 本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。
- 最適化リンケージエディタのメッセージについては，lnkcmd オプションにより，最適化リンケージエディタの message オプションおよび nomessage オプションを指定することで出力制御が可能です。

-nomessage

<コンパイル・オプション / ソースオプション>

[指定形式]

```
-nomessage[= <エラー番号> [- <エラー番号>][,…]
```

[詳細説明]

- nomessage オプションを指定した場合，インフォメーションレベル・メッセージの出力を抑止します。
- サブオプションでエラー番号を指定すると，指定したインフォメーションレベル・メッセージの出力だけを抑止します。
- エラー番号が複数ある場合にはカンマ（,）で区切って指定することができます。
- <エラー番号>-<エラー番号>のようにハイフン（-）で抑止するエラー番号の範囲を指定することもできます。
- エラー番号には、インフォメーションを表す「M」から始まるメッセージ番号の、末尾 (右側) の 5 桁を指定してください。

  例) インフォメーションメッセージ M0523009 の場合

  -nomessage=23009

[備考]

- 本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。
- 最適化リンケージエディタのメッセージについては，lnkcmd オプションにより，最適化リンケージエディタの message オプションおよび nomessage オプションを指定することで出力制御が可能です。
- nomessage オプションを複数回指定した場合，指定したすべてのエラー番号について抑止します。
- 本オプションは、メッセージの番号 (コンポーネント番号を含む) が 0510000 ～ 0549999 であるもののみ対象とします。
- 番号が 0520000 ～ 0529999 以外の警告メッセージは、本オプションに番号を指定しても抑止できません。同時に -chagne_message を用いてインフォメーションに変更してください。

-change_message

<コンパイル・オプション / ソースオプション>

[指定形式]

```
-change_message= <sub>[,…]
                    <sub> : < エラーレベル >[=< エラー番号 >[- < エラー番号 >][,…]]
                            < エラーレベル > : { information | warning | error }
```

[詳細説明]

- インフォメーション，ウォーニングのメッセージ・レベルを変更します。
- エラー番号が複数ある場合にはカンマ（,）で区切って指定することができます。
- エラー番号には、インフォメーションを表す「M」または警告を表す「W」から始まるメッセージ番号の、末尾 (右側) の 5 桁を指定してください。
  例) インフォメーションメッセージ M0523009 の場合
  -change_message=error=23009

[例]

```
change_message=information= エラー番号
```

- ウォーニングレベルの指定エラー番号のみインフォメーションレベルに変更します。

```
change_message=warning= エラー番号
```

- インフォメーションレベルの指定エラー番号のみウォーニングレベルに変更します。

```
change_message=error= エラー番号
```

- インフォメーション，ウォーニングレベルの指定エラー番号のみエラーレベルに変更します。

```
change_message=information
```

- すべてのウォーニングメッセージをインフォメーションレベルに変更します。

```
change_message=warning
```

- すべてのインフォメーションメッセージをウォーニングレベルに変更します。

```
change_message=error
```

- すべてのインフォメーション，ウォーニングメッセージをエラーレベルに変更します。

[備考]

- インフォメーションレベルに変更したメッセージについては，nomessage オプション指定により出力を抑止できます。
- 本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。
- 最適化リンケージエディタのメッセージについては，lnkcmd オプションにより，最適化リンケージエディタの message オプションおよび nomessage オプションを指定することで出力制御が可能です。
- 本オプションを複数回指定した場合，指定したすべてのエラー番号について有効になります。
- 警告メッセージおよびインフォメーションメッセージ以外のメッセージは対象外です。本オプションに指定しても無視します。
- misra2004 オプション指定時に表示する，MISRA 検出メッセージ（記号 (M) を表示）は本オプション制御対象外です。
- 一部のメッセージでエラー (E) や警告 (W) などのメッセージ種別が変化することがあります。メッセージ種別が変わっても、番号 (コンポーネント番号およびメッセージ番号) で、メッセージの意味は変わりません。

-file_inline_path

<コンパイル・オプション / ソースオプション>

［指定形式］

```
-file_inline_path= <パス名>[,…]
```

［詳細説明］

- V2.00 で、本オプションは無効になりました。指定した場合、無視されますが、従来バージョンとの互換性のためエラーにはなりません。

-comment

<コンパイル・オプション / ソースオプション>

[指定形式]

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

- 省略時解釈

  comment=nonest です。

[詳細説明]

- comment=nest を指定した場合，ネストしたコメントの記述を可能にします。
- comment=nonest を指定した場合，ネストしたコメントを記述するとエラーになります。

[例]

- comment=nest を指定した場合はすべてコメントと解釈しますが，comment=nonest を指定した場合は [1] でコメントが終わっていると解釈します。

```
/* This is an example of /* nested */ comment */
                                    [1]
```

-check

<コンパイル・オプション / ソースオプション>

[指定形式]

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

[詳細説明]

- R8C, M16C ファミリ用 C/C++ コンパイラ，H8, H8S, H8SX ファミリ用 C/C++ コンパイラおよび SuperH ファミリ用 C/C++ コンパイラ向けにコーディングした C/C++ 言語ソースファイルを本コンパイラへ流用する際，互換性に影響するオプション指定，ソース記述をチェックすることができます。
- check=nc では，R8C,M16C ファミリ用 C コンパイラとの互換性をチェックします。
  チェックされるオプションや型には，次のようなものがあります。
  - オプション：signed_char, signed_bitfield, bit_order=left, endian=big, dbl_size=4
  - inline，enum 型，#pragma BITADDRESS，#pragma ROM，#pragma PARAMETER，asm()
  - -int_to_short の指定がないときに，signed short 範囲外の定数を int,signed int 型へ代入，あるいは unsigned short 範囲外の定数を int 型または unsigned int 型へ代入
  - signed short, unsigned short 共範囲外の定数を long, long long 型へ代入
  - signed short 範囲外の定数と int, short, char 型（char 型は符号付き除く）との比較式
- check=ch38 では，H8,H8S,H8S ファミリ用 C/C++ コンパイラとの互換性をチェックします。
  チェックされるオプションや型には，次のようなものがあります。
  - オプション：unsigned_char, unsigned_bitfield, bit_order=right, endian=little, dbl_size=4
  - __asm，#pragma unpack
  - signed long の最大値より大きな定数との比較式
  - -int_to_short の指定がないときに，signed short 範囲外の定数を int,signed int 型へ代入，あるいは unsigned short 範囲外の定数を int 型または unsigned int 型へ代入
  - signed short, unsigned short 共範囲外の定数を long, long long 型へ代入
  - signed short 範囲外の定数と int, short, char 型（char 型は符号付き除く）との比較式
- check=shc では，SuperH ファミリ用 C/C++ コンパイラとの互換性をチェックします。
  チェックされるオプションや型には，次のようなものがあります。
  - オプション：unsigned_char, unsigned_bitfield, bit_order=right, endian=little, dbl_size=4 , round=nearest
  - #pragma unpack
  - volatile 修飾した変数
- 表示された項目により，それぞれ次の項目を確認ください。
  - オプション：言語仕様で規定されていない実装依存の内容がコンパイラ間で異なっています。メッセージで出力されたオプションの選択を確認してください。
  - 拡張仕様：プログラムの動作に影響を及ぼす可能性がある拡張仕様です。メッセージで出力された拡張仕様の記述を確認してください。

［備考］

- dbl_size=4 が有効な時に，R8C, M16C ファミリ用 C コンパイラ，H8, H8S, H8SX ファミリ用 C/C++ コンパイラおよび SuperH ファミリ用 C/C++ コンパイラと浮動小数点関連の変換 / ライブラリの計算結果が異なる場合があります。
  dbl_size=4 は，本コンパイラでは，double 型および long double 型を 32 ビットにしますが，各種 R8C, M16C ファミリ用 C コンパイラ（fdouble_32），H8, H8S, H8SX ファミリ用 C/C++ コンパイラ（double=float）および SuperH ファミリ用 C/C++ コンパイラ（double=float）では，double 型のみ 32 ビットにします。
- unsigned int 型と long 型をオペランドとする二項演算（加減乗除や比較など）に対する結果が，SuperH ファミリ用 C/C++ コンパイラと異なる場合があります。
  本コンパイラではオペランドを unsigned long 型に変換してから演算しますが，SuperH ファミリ用 C/C++ コンパイラ（ただし，strict_ansi を指定しないとき）では，signed long long 型に変換してから演算します。
- volatile 修飾した変数に対し，読み出しや書き込みのサイズが，SuperH ファミリ用 C/C++ コンパイラと異なる場合があります。
  volatile 修飾したビットフィールドは，本コンパイラでは宣言型より小さなサイズでアクセスすることがありますが，SuperH ファミリ用 C/C++ コンパイラでは宣言型のサイズ通りにアクセスします。
- 構造体およびビットフィールドメンバの割り付けについては，本オプションでメッセージを出力しません。割り付けを意識した宣言をしている場合には，RX コーディング編の「3.1.4 データの内部表現と領域」の項目を参照してください。
- R8C, M16C ファミリ用 C コンパイラ（fextend_to_int を指定しない）では，条件式で汎整数拡張を行わずに評価したコードを生成するので，本コンパイラの生成コードと動作が異なる場合があります。

-misra2004

<コンパイル・オプション / ソースオプション>

[指定形式]

```
-misra2004= {
              all
              | apply=< ルール番号 >[,< ルール番号 >,...]
              | ignore=< ルール番号 > [,< ルール番号 >,...]
              | required
              | required_add=< ルール番号 >[,< ルール番号 >,...]
              | required_remove=< ルール番号 >[,< ルール番号 >,...]
              | < ファイル名 >}
```

[詳細説明]

- MISRA-C:2004 のルールチェック機能を有効にし，そのチェック対象を指定します。
- -misra2004=all オプション指定時は，サポートしているすべてのルールをチェック対象とします。
- -misra2004=apply< ルール番号 >[,< ルール番号 >,...] オプション指定時は，サポートしているルールのうち，指定されたルール番号をチェック対象とします。
- -misra2004=ignore=< ルール番号 >[,< ルール番号 >,...] オプション指定時は，サポートしているルールのうち，指定されたルール番号以外のルールをチェック対象とします。
- -misra2004=required オプション指定時は，サポートしているルールのうち，ルールの分類が“required”になっているルールをチェック対象とします。
- -misra2004=required_add=< ルール番号 >[,< ルール番号 >,...] オプション指定時は，サポートしているルールのうち，ルールの分類が“required”になっているルールと指定されたルール番号をチェック対象とします。
- -misra2004=required_remove=< ルール番号 >[,< ルール番号 >,...] オプション指定時は，サポートしているルールのうち，ルールの分類が“required”になっているルールから指定されたルール番号を除いたルール番号をチェック対象とします。
- -misra2004 =< ファイル名 > オプション指定時は，サポートしているルールのうち，指定されたファイル名に記載されたルール番号をチェック対象とします。ファイル名で指定されたファイル内の記述は，1 ルールを 1 行で記述します。
- ルール番号は，10 進数値およびピリオド（.）で指定してください。
- MISRA-C:2004 のチェック項目に該当した場合，次の形式でメッセージを表示します。

  ファイル名（*行番号*）: M0523028 Rule *ルール番号*: *メッセージ*
- -misra2004=< ファイル名 > の指定を複数回行った場合は、最後に指定したファイルのみ有効になります。

［備考］

- -misra2004 オプションは複数回指定できます。ただし、指定された種類 (-misra2004= 以降に指定する名前) が複数ある場合は、最後に記述した種類と同じものだけを有効とします。

  (指定例)

  ... -misra2004=ignore=2.2 -misra2004=apply=2.3

  -misra2004=required_add=4.1 -misra2004=apply=4.2

  -misra2004=apply=5.2 ...

  ignore, apply, required_add の 3 種類の -misra2004 が同時に指定されていますが、最後に連続して指定した 2 つの apply の指定のみが有効になります。その結果、ルール 4.2 と 5.2 がチェック対象になります。
- サポートしていないルール番号を，各オプションの < ルール番号 > に指定した場合はエラー C0523031 となり，そこで処理を中止します。
- -misra2004=< ファイル名 > で指定したルールファイルがオープンできない場合，エラー C0523029 となります。また，ルールファイルからルール番号が取り出せない場合は，エラー C0523030 となり，いずれの場合もそこで処理を中止します。
- -lang オプションで cpp,c99 または ecpp が選択された場合，本オプションを無視します。
- -output=prep と同時に指定した場合，本オプションを無視します。
- 本オプションでサポートしている MISRA-C:2004 のルール番号を次に示します。

  [required であるルール番号]

  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

  [required ではないルール番号]

  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

- #pragma などの拡張機能を用いたソースでは，ルールチェックの一部が抑止されます。
  抑止される内容は，check_language_extension オプションの項目を参照してください。
- misra2004 オプションで表示される MISRA 診断メッセージは，change_message オプションで制御することはできません。

-ignore_files_misra

<コンパイル・オプション / ソースオプション>

[指定形式]

```
-ignore_files_misra=< ファイル名 >[,< ファイル名 >,...]
```

[詳細説明]

- MISRA-C:2004 のルールチェック対象外のソースファイルを指定します。

[備考]

- コマンドライン上で同一オプションを複数回指定した場合には，それぞれのオプションが有効になります。
- -misra2004 オプションの指定がない場合は，本オプションを無視します。
- 指定されたソースファイル名が，コンパイル対象でない場合は，指定されたソースファイル名を無視します。

-check_language_extension

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-check_language_extension
```

［詳細説明］

- C/C++ 言語規格から独自に拡張した言語仕様が有効な場合に一部抑止される，MISRA2004 ルールチェックを有効にします。
- 本コンパイラでは，misra2004 オプションのデフォルトでは，次の場合はチェックが抑止されます。これをチェックしたい場合は，misra2004 オプション指定時に，check_language_extension オプションを指定してください。
  - プロトタイプ宣言がない場合（ルール 8.1）で，該当関数に #pragma entry または #pragma interrupt のいずれかの指定があるとき。

［例］

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

- 関数 vfunc にはプロトタイプ宣言がありませんが，#pragma interrupt の対象なので -misra2004=all を指定してコンパイルしても，-check_language_extension の指定がないと，ルール 8.1 のチェックメッセージが表示されません。

［備考］

- -misra2004 オプションの指定がない場合は，本オプションを無視します。

オブジェクトオプション

<コンパイル・オプション / オブジェクトオプション>

オブジェクトオプションには，次のものがあります。

-output

<コンパイル・オプション / オブジェクトオプション>

［指定形式］

```
-output = <sub> [=< ファイル名 >]
        <sub> : { prep | src | obj | abs | hex | sty }
```

- 省略時解釈

  output=obj です。

［詳細説明］

- 出力ファイルの形式を指定します。
- サブオプションと出力ファイルの一覧を以下に示します。
- < ファイル名 > を指定しない場合は，先頭に入力したソースファイル名に以下表の拡張子をつけたファイルを作成します。

表 A—3　サブオプション出力形式

サブオプション	出力形式	ファイル名指定省略時の拡張子
prep	プリプロセッサ展開後のソースファイル	C（C89, C99）言語ソースファイル：p C++ 言語ソースファイル：pp
src	アセンブリ・ソース・ファイル	src
obj	リロケータブル・ファイル	obj

サブオプション	出力形式	ファイル名指定省略時の拡張子
abs	アブソリュート・ファイル	abs
hex	インテル HEX 形式ファイル	hex
sty	モトローラ S 形式ファイル	mot

注 リロケータブルファイルは，アセンブラの出力ファイルです。
アブソリュートファイル，インテル HEX 形式ファイル，およびモトローラ S 形式ファイルは，最適化リンケージエディタの出力ファイルです。

［備考］

- 指定した形式のファイルを作成するための中間ファイルは，フォルダ指定があればそのフォルダに，フォルダ指定がなければカレントフォルダに作成します。

-noline

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-noline
```

[詳細説明]

- プリプロセッサ展開時に #line 出力を抑止します。

[備考]

- 本オプションは output=prep オプションの指定が無い場合は無効となります。

-debug

<コンパイル・オプション / オブジェクトオプション>

［指定形式］

```
-debug
```

［詳細説明］

- debug オプションを指定した場合，C ソース・レベル・デバッグに必要なデバッグ情報を出力します。
- debug オプションは，最適化オプションを指定した場合も有効となります。

-nodebug

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-nodebug
```

[詳細説明]

- nodebug オプションを指定した場合，デバッグ情報を出力しません。

-section

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-section = <sub>[,…]
           <sub>: { P = < セクション名 > |
                    C = < セクション名 > |
                    D = < セクション名 > |
                    B = < セクション名 > |
                    L = < セクション名 > |
                    W = < セクション名 > }
```

- 省略時解釈

  section=P=P,C=C,D=D,B=B,L=L,W=W です。

[詳細説明]

- セクション名を指定します。
  - section=P =< セクション名 > は，プログラム領域のセクション名を指定します。
  - section=C =< セクション名 > は，定数領域のセクション名を指定します。
  - section=D =< セクション名 > は，初期化データ領域のセクション名を指定します。
  - section=B =< セクション名 > は，未初期化データ領域のセクション名を指定します。
  - section=L =< セクション名 > は，リテラル領域のセクション名を指定します。
  - section=W=< セクション名 > は，switch 文分岐テーブル領域のセクション名を指定します。
- < セクション名 > は，英字，数字，下線（_），または $ の列で，先頭が数字以外のものです。

[備考]

- V.1.00 と同様に L セクションを出力せず，リテラル領域を C セクションに出力したい場合は，section=L=C を選択してください。
- L セクションを C セクションと同じ名前のセクション名に変更する場合を除き，領域が異なるセクションに同じセクション名を指定できません。
- セクション名長の翻訳限界については，「 翻訳限界」を参照してください。

-stuff

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-stuff
```

[詳細説明]

- stuff オプションを指定した場合，すべての変数をアライメント数に応じてアライメント数が 4 のセクション，2 のセクション，1 のセクションに配置します（表 B-3）。

表 A—4　stuff オプション指定時の，各変数と出力先セクションの関係

変数の種類	変数のアライメント数	変数が所属するセクション
const 修飾変数	4	C
	2	C_2
	1	C_1
初期値あり変数	4	D
	2	D_2
	1	D_1
初期値なし変数	4	B
	2	B_2
	1	B_1
switch 文分岐テーブル	4	W
	2	W_2
	1	W_1

- C，D，B は section オプションまたは #pragma section で指定したセクション名になります。
- W は section オプションで指定したセクション名になります。C セクション内で初期値がない変数が初期値のある変数の後に出力されることを除き，各セクション内のデータは定義順に出力されます。

［例］

```
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
```

［備考］

- sutff オプションは B,D,C および W 以外のセクションに対しては無効です。

-nostuff

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-nostuff [= <セクション種別>[,…]]
           <セクション種別>:{ B | D | C | W}
```

[詳細説明]

- nostuff オプションを指定した場合，指定した <セクション種別> に属する変数をアライメント数が 4 のセクションに配置します。
- <セクション種別> を省略した場合は，すべてのセクション種別の変数が対象になります。
- C，D，B は section オプションまたは #pragma section で指定したセクション名になります。
- W は section オプションで指定したセクション名になります。
- C セクション内で初期値がない変数が初期値のある変数の後に出力されることを除き，各セクション内のデータは定義順に出力されます。

[例]

```
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
```

[備考]

- nostuff オプションに B,D,C および W 以外のセクションを指定することはできません。

-instalign4

<コンパイル・オプション / オブジェクトオプション>

［指定形式］

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

- 省略時解釈

  関数先頭，switch 文の case および default ラベルを 4 バイトで配置アドレスを命令実行向け整合します。

［詳細説明］

- 分岐先の命令実行向け整合を行います。
- instalign4 を指定した場合は 4 バイト配置アドレスを命令実行向け整合します。
- 命令実行向け整合とは，-instalign4 のサブオプションで選択した種類の分岐先の命令が，アライメント数（4）の倍数であるアドレスをまたいでいる場合（*1）にだけ整合を取るものです。
- 分岐先の種類は，-instalign4 のサブオプションの指定により次の 3 種類から選択します（*2）。
  - 指定なし : 関数先頭，switch 文の case および default ラベル
  - inmostloop: 各最内周ループの先頭，関数先頭，switch 文の case および default ラベル
  - loop: 各ループの先頭，関数先頭，switch 文の case および default ラベル
- 本オプションを選択すると，プログラムセクションのアライメント数が 1 から 4 に変わります。
- 本オプションは，分岐先命令のアドレス整合することで RX CPU の命令キューを効率よく動作させ，プログラムの実行速度向上を図るものです。
- 次の用途を想定した仕様となっております。
  - instalign4 … 命令キューが 32 ビットの CPU（主に RX200 シリーズ）で速度向上を図る場合

注 1. 命令サイズがアライメント数以下の場合です。命令サイズがアライメント数よりも大きい場合は，またいでいる箇所が 2 以上の場合にだけ整合を取ります。

2. ここに挙げたもの以外の分岐先は，命令実行向け整合の対象ではありません。たとえば，loop を選択した場合はループの先頭は対象ですが，ループ内にあるループを構成しない if 文などの分岐先は対象ではありません。

-instalign8

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

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

- 省略時解釈

  関数先頭，switch 文の case および default ラベルを 8 バイトで配置アドレスを命令実行向け整合します。

[詳細説明]

- 分岐先の命令実行向け整合を行います。
- instalign8 を指定した場合は 8 バイトでそれぞれ配置アドレスを命令実行向け整合します。
- 命令実行向け整合とは，-instalign8 のサブオプションで選択した種類の分岐先の命令が，アライメント数（8）の倍数であるアドレスをまたいでいる場合（*1）にだけ整合を取るものです。
- 分岐先の種類は，-instalign8 のサブオプションの指定により次の 3 種類から選択します（*2）。
  - 指定なし : 関数先頭，switch 文の case および default ラベル
  - inmostloop: 各最内周ループの先頭，関数先頭，switch 文の case および default ラベル
  - loop: 各ループの先頭，関数先頭，switch 文の case および default ラベル
- 本オプションを選択すると，プログラムセクションのアライメント数が 1 から 8 に変わります。
- 本オプションは，分岐先命令のアドレス整合することで RX CPU の命令キューを効率よく動作させ，プログラムの実行速度向上を図るものです。
- 次の用途を想定した仕様となっております。
  - instalign8 … 命令キューが 64 ビットの CPU（主に RX600 シリーズ）で速度向上を図る場合

注 1. 命令サイズがアライメント数以下の場合です。命令サイズがアライメント数よりも大きい場合は，またいでいる箇所が 2 以上の場合にだけ整合を取ります。

2. ここに挙げたもの以外の分岐先は，命令実行向け整合の対象ではありません。たとえば，loop を選択した場合はループの先頭は対象ですが，ループ内にあるループを構成しない if 文などの分岐先は対象ではありません。

［例］

- ＜C ソース＞

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

- ＜出力コード＞

[-instalign8 を指定してコンパイルした場合]

下記は，各関数の先頭を，8 バイトで命令実行向け整合させた場合の例です。

8 バイトの命令実行向け整合では，対象命令が 8 バイトの境界をまたがない限り，配置アドレスを変更しないため，実際に整合がかかるのは関数 f2 のアドレスだけです。

```
    .SECTION P,CODE,ALIGN=8
    .INSTALIGN 8
_f1:                        ; 関数 f1。配置アドレス =0000H
    ADD     #01H,R1         ; 2 バイト
    RTS                     ; 1 バイト
    .INSTALIGN 8
_f2:                        ; 関数 f2。配置アドレス =0008H ※整合有り
                            ; 0003H に 6 バイト命令が来ると，8 バイト境界を
                            ; またぐため，整合が取られる。
    MOV.L   #_a,R4          ; 6 バイト
    MOV.L   #0,[R4]         ; 3 バイト
    RTS                     ; 1 バイト
    .INSTALIGN 8
_f3:                        ; 関数 f3。配置アドレス =0012H
     RTS
    .END
```

-noinstalign

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-noinstalign
```

[詳細説明]

- 分岐先の命令実行向け整合を行いません。

-nouse_div_inst

<コンパイル・オプション / オブジェクトオプション>

[指定形式]

```
-nouse_div_inst
```

[詳細説明]

- プログラム中のすべての除算および剰余算を，DIV 命令，DIVU 命令および FDIV 命令を使わないコードを生成します。

[備考]

- 本オプション指定時は，DIV,DIVU および FDIV 命令を生成する代わりに，それぞれの命令に相当する処理を行うランタイム関数の呼び出しに置き換えます。
- このため，ROM サイズやコード実行速度といったコード効率が悪くなる場合があります。

リストオプション

＜コンパイル・オプション / リストオプション＞

リストオプションには，次のものがあります。

-listfile

＜コンパイル・オプション / リストオプション＞

［指定形式］

```
-listfile[={< ファイル名 >|< パス名 >}]
```

- 省略時解釈

  ソースファイルと同じファイル名で，拡張子が lst のソースリストファイルを作成します。

［詳細説明］

- listfile オプションを指定した場合，ソースリストファイルを出力します。＜ファイル名＞を指定することもできます。
- < ファイル名 > の代わりに、あらかじめ存在するフォルダを < パス名 > として指定することもできます。この場合は、ソースリストファイル名としてコンパイルまたはアセンブル対象のソースファイルの拡張子を .lst に変更したものを、< パス名 > に選択したフォルダに出力します。

［備考］

- 本オプションを指定してリンケージリストを出力することはできません。
- リンケージリストを出力するには，lnkcmd オプションにより最適化リンケージエディタの list オプションを指定してください。
- コンパイラが出力する情報は，ソースリストに書き込まれます。
- ソース・リスト・ファイルのフォーマットについては，「アセンブリ・リスト・ファイル」を参照ください。
- < パス名 > を使用する場合は、パス名に該当するフォルダはあらかじめ作成しておいてください。存在しない場合は、listfile には < パス名 > の代わりに < ファイル名 > が選択されたものとみなします。

-nolistfile

<コンパイル・オプション / リストオプション>

［指定形式］

```
-nolistfile
```

［詳細説明］

- ソースリストファイルは出力しません。

-show

<コンパイル・オプション / リストオプション>

[指定形式]

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

[詳細説明]

- ソースリストファイルの内容の設定を行います。
- サブオプションと指定内容の一覧を以下に示します。

表 A—5　サブオプション指定一覧

サブオプション	内容
source	C/C++ ソースを出力
conditionals	条件アセンブルで条件が偽となる行も含めて出力
definitions	.DEFINE を置き換える以前の情報を出力
expansions	アセンブラマクロ記述展開行を出力

[備考]

- 本オプションは listfile オプション指定時のみ有効です。
- コンパイラが出力する情報は，ソースリストに書き込まれます。ソースリストファイルのフォーマットについては，「アセンブリ・リスト・ファイル」を参照ください。

最適化オプション

<コンパイル・オプション / 最適化オプション>

最適化オプションには，次のものがあります。

- -optimize
- -goptimize
- -speed
- -size
- -loop
- -inline
- -noinline
- -file_inline（※）無効オプション
- -case
- -volatile
- -novolatile
- -const_copy
- -noconst_copy
- -const_div
- -noconst_div
- -library
- -scope
- -noscope
- -schedule
- -noschedule
- -map
- -smap
- -nomap
- -approxdiv
- -enable_register（※）無効オプション
- -simple_float_conv
- -fpu
- -nofpu
- -alias
- -float_order（※）無効オプション
- -ip_optimize
- -merge_files
- -whole_program

-optimize

<コンパイル・オプション / 最適化オプション>

[指定形式]

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

- 省略時解釈

  -optimize=2 です。

[詳細説明]

- 最適化レベルを指定します。
- optimize=0 を指定した場合，最適化を実施しません。これにより，デバッグ情報を高い精度で出力でき，ソースレベルデバッグがしやすくなります。
- optimize=1 を指定した場合，自動変数のレジスタ割付，関数出口ブロックの統合，統合可能な複数命令の統合など，一部最適化を実施します。これにより，optimize=0 指定時よりもコードサイズを削減できます。
- optimize=2 を指定した場合，全般的に最適化を実施します。ただし，実施する最適化の内容は，size/speed オプションの選択によって異なります。
- optimize=max を指定した場合，実施可能な最適化を最大限に行います。たとえば，最適化の適用範囲を最大限に拡大したり，speed オプション指定時には，大規模なループ展開を可能にします。最適化の効果が期待できる反面，コンパイル時間の増大や，speed オプション指定時のコードサイズの大幅な増加など，副作用を伴う場合があります。

[備考]

- 各種最適化オプションの説明で，デフォルトが記述されていないものは，optimize オプションと speed, size オプションの指定値によりデフォルトが変化することを意味します。
- デフォルトについての詳細は，speed, size オプションを参照ください。

-goptimize

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-goptimize
```

［詳細説明］

- モジュール間最適化時に使用する付加情報を，出力ファイル内部に生成します。
- 本オプションを指定したファイルは，リンク時にモジュール間最適化の対象になります。

-speed

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-speed
```

［詳細説明］

- speed オプションを指定した場合，実行性能重視の最適化を実施します。

［備考］

- speed オプションを指定した場合，optimize オプションの指定により，以下オプションが指定されているとみなします。

＜ optimize=max 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=8	inline=250	const_div	schedule	const_copy	noscope	map 注 nomap 注	alias=ansi

注　入力が C/C++ ソースで，かつ出力の指定が output=abs か mot の場合は map がデフォルトに，それ以外では nomap がデフォルトとなります。

＜ optimize=2 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=2	inline=100	const_div	schedule	const_copy	scope	nomap	alias=noansi

＜ optimize=0 または optimize=1 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=1	noinline	const_div	noschedule	noconst_copy	scope	nomap	alias=noansi

-size

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-size
```

［詳細説明］

- size オプションを指定した場合，コードサイズ重視の最適化を実施します。

［備考］

- size オプションを指定した場合，optimize オプションの指定により，以下オプションが指定されているとみなします。ただし，以下オプションを明示的に指定した場合は指定したオプションが有効になります。

＜ optimize=max 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
size	loop=1	inline=0	noconst_div	schedule	const_copy	noscope	map 注 nomap 注	alias=ansi

注　入力が C/C++ ソースで，かつ出力の指定が output=abs か mot の場合は map がデフォルトに，それ以外では nomap がデフォルトとなります。

＜ optimize=2 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
size	loop=1	noinline	noconst_div	schedule	const_copy	scope	nomap	alias=noansi

＜ optimize=0 または optimize=1 指定時＞

	ループ展開	インライン展開	定数除算の乗算化	命令並び換え	const 修飾変数の定数伝播	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
size	loop=1	noinline	noconst_div	noschedule	noconst_copy	scope	nomap	alias=noansi

-loop

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-loop[=< 数値 >]
```

- 省略時解釈

  loop=2 です。

［詳細説明］

- ループ展開の最適化を行うかどうかを指定します。
- loop オプションを指定した場合，ループ文（for, while, do-while）を展開します。
- < 数値 > で，最大で何倍の展開を行うかを指定することができます。< 数値 > は 1 ～ 32 の整数を指定することができます。< 数値 > を指定しなかった場合は 2 とします。
- 本オプションの省略時解釈は，optimize オプションと speed, size オプションの指定に従います。詳細は，speed, size オプションを参照してください。

［備考］

- optimize=0 または optimize=1 のときは本オプションの指定は無効です。

-inline

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-inline[=< 数値 >]
```

- 省略時解釈

  inline=100 です。

[詳細説明]

- 関数の自動インライン展開を行います。
- < 数値 > として使える値の範囲は 0 ～ 65535 です。
- inline オプションを指定した場合，自動インライン展開を行います。ただし，#pragma noinline を指定した関数はインライン展開を行いません。
- < 数値 > で，関数サイズが何 % 増加するまでインライン展開を行うかを指定できます。例えば，inline=100 を指定した場合，関数サイズが 100% 増加するまで（2 倍まで）インライン展開を行います。
- 本オプションの省略時解釈は，optimize オプションと speed, size オプションの指定に従います。詳細は，speed, size オプションを参照してください。

[備考]

- #pragma inline を指定した関数，および inline 指定子付きの関数は，オプションの指定に関わらず，展開を試みます。
- 確実に関数をインライン展開したい場合は，#pragma inline を関数に指定してください。
- 本オプションの選択もしくは inline 指定子を関数に指定しても，コンパイラで効率が悪くなると判断したときは，インライン展開を行わないことがあります。

-noinline

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-noinline
```

[詳細説明]

- 関数の自動インライン展開を行いません。

[備考]

- #pragma inline を指定した関数，および inline 指定子付きの関数は，オプションの指定に関わらず，展開を試みます。

-file_inline

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-file_inline=< ファイル名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- V2.00 で、本オプションは無効になりました。指定した場合、無視されますが、従来バージョンとの互換性のためエラーにはなりません。

［備考］

- C(C99) ソースの場合は、代替機能である -merge_files オプションが使用できます。-file_inline に指定していたファイル (-file_inline_path を併用していた場合は、パス名を含めて) を、ソースファイルのひとつとして追加してください。
- -merge_files 機能には、いくつか注意点があります。-merge_files オプションの [備考] 欄も参照してください。

-case

<コンパイル・オプション / 最適化オプション>

[指定形式]

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

- 省略時解釈

  case=auto です。

[詳細説明]

- switch 文のコード展開方式を指定します。
- case=ifthen を指定した場合，switch 文を if_then 方式で展開します。if_then 方式は，switch 文の評価式の値と case ラベルの値を比較し，一致すれば case ラベルの文へ飛ぶ処理を case ラベルの回数繰り返す展開方式です。この方式は，switch 文に含まれる case ラベルの数に比例してオブジェクトコードのサイズが増大します。
- case=table を指定した場合，switch 文をテーブル方式で展開します。テーブル方式は，case ラベルの飛び先を分岐テーブルに確保し，1 回の分岐テーブルの参照で switch 文の評価式と一致する case ラベルの文へ飛ぶ展開方式です。この方式は，switch 文に含まれる case ラベルの数に比例して分岐テーブルのサイズが増えますが，実行速度は常に一定です。分岐テーブルは，switch 文分岐テーブル領域のセクションに出力されます。
- case=auto を指定した場合，if_then 方式，テーブル方式いずれかをコンパイラが自動的に選択します。

[備考]

- case=table 指定時に作成される分岐テーブルは，nostuff オプション指定時は W セクションに出力されますが，nostuff オプション指定がない場合は，switch 文の規模により W，W_2 または W_1 セクションのいずれかに振り分けて出力されます。

-volatile

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-volatile
```

［詳細説明］

- volatile を指定した場合，すべての外部変数を volatile 宣言したものとして扱います。したがって，外部変数のアクセス回数，アクセス順序は C/C++ 言語ソースファイルで記述した通りになります。

-novolatile

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-novolatile
```

［詳細説明］

- novolatile を指定した場合，volatile 修飾のない外部変数に対して最適化を行います。したがって，外部変数のアクセス回数，アクセス順序が C/C++ 言語ソースファイルで記述した場合と異なることがあります。

-const_copy

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-const_copy
```

- 省略時解釈

  optimize=2 または optimize=max オプションを指定した場合は const_copy です。

[詳細説明]

- const_copy を指定した場合，const 修飾型外部変数についても定数伝播を行います。

[備考]

- C++ 言語ソースファイルの const 修飾型変数については，本オプションで制御することはできません（常に定数伝播されます）。

-noconst_copy

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noconst_copy
```

- 省略時解釈

  本オプションの省略時解釈は，optimize=1 または optimize=0 オプションを指定した場合は noconst_copy です。

［詳細説明］

- noconst_copy を指定した場合，const 修飾型外部変数の定数伝播を抑止します。

［備考］

- C++ 言語ソースファイルの const 修飾型変数については，本オプションで制御することはできません（常に定数伝播されます）。

-const_div

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-const_div
```

- 省略時解釈

  speed オプションを指定した場合は const_div です。

［詳細説明］

- const_div を指定した場合，ソースファイル中の整数型定数による除算および剰余算を，乗算やビット演算 (シフトやビット論理積) を用いた命令列に変換します。

［備考］

- シフト演算のみで行える定数乗算，およびビット論理積のみで行える剰余算は，const_div オプションの制御対象外となります。

-noconst_div

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noconst_div
```

- 省略時解釈

  size オプションを指定した場合は noconst_div です。

［詳細説明］

- noconst_div を指定した場合，ソースファイル中の整数型定数による除算および剰余算に、それぞれに対応する除算命令および剰余算命令 (2 のべき乗定数値による符号なし整数の除算および剰余算を除く) を用います。

［備考］

- シフト演算のみで行える定数乗算，およびビット論理積のみで行える剰余算は，noconst_div オプションの制御対象外となります。

-library

<コンパイル・オプション / 最適化オプション>

[指定形式]

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

- 省略時解釈

  library=intrinsic です。

[詳細説明]

- library=function を指定した場合，ライブラリ関数をすべて関数呼び出しします。
- library=intrinsic を指定した場合，abs()，fabsf() およびストリング操作命令が使用できるライブラリ関数を命令展開します。

[備考]

- library=intrinsic とともに -isa=rxv2 が指定されている場合は、sqrtf 関数または -dbl_size=4 指定時の sqrt 関数のそれぞれの呼び出しを FSQRT 命令に置き換えます。ただし、FSQRT 命令に展開された場合は、変数 errno の内容を変更しません。

-scope

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-scope
```

- 省略時解釈

  optimize=max オプションを指定した場合以外は scope です。

［詳細説明］

- scope を指定した場合，サイズの大きい関数について，最適化範囲を複数に分割してコンパイルします。
- 本オプションは，プログラムによって実行性能に影響しますので，性能チューニング時に試してください。

-noscope

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noscope
```

- 省略時解釈

  optimize=max オプションを指定した場合は noscope です。

［詳細説明］

- noscope を指定した場合，最適化範囲を分割せずにコンパイルします。最適化範囲が広がることによりコンパイル速度は遅くなりますが，一般的にはオブジェクト性能が向上します。ただし，レジスタ数が不足するとオブジェクト性能が低下する場合があります。
- 本オプションは，プログラムによって実行性能に影響しますので，性能チューニング時に試してください。

-schedule

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-schedule
```

- 省略時解釈

  optimize=2 または optimize=max オプションを指定した場合は schedule です。

［詳細説明］

- schedule を指定した場合，パイプライン処理を考慮した命令並べ替えを行います。

-noschedule

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noschedule
```

- 省略時解釈

  optimize=1 または optimize=0 オプションを指定した場合は noschedule です。

［詳細説明］

- noschedule を指定した場合，命令並べ替えを行いません。基本的に C/C++ 言語ソースファイルで記述した順番で処理を行います。

-map

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-map[=< ファイル名 >]
```

- 省略時解釈

  optimize=max オプションを指定した場合は map です。

［詳細説明］

- 外部変数アクセス最適化を行います。
- 最適化リンケージエディタが生成する外部シンボル割り付け情報を元にベースアドレスを設定し，外部変数もしくは静的変数のアクセスをベースアドレス相対で行うコードを生成します。
- map オプションによる外部変数アクセス最適化を使用する場合は，output オプションの指定により使い方が異なります。
- [output=abs または mot の場合]

  map のみ指定してください（optimize=max 指定時は不要）。自動的にコンパイル・リンクを 2 回行い，外部シンボル割り付け情報を元にベースアドレスを設定したコード生成を行います。
- [output=obj の場合]

  ソースファイルを本オプションを指定しないで一度コンパイルし，最適化リンケージエディタでのリンク時に map=< ファイル名 > を指定して外部シンボル割り付け情報ファイルを作成し，再度 ccrx に map=< ファイル名 > を指定してコンパイルしてください。

［例］

- ＜C ソース＞

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

- ＜出力コード＞

```
_func:
    MOV.L       #_A,R4      ; A のアドレスをベースアドレスに設定
    MOV.L       #1,[R4]
    MOV.L       #2,4[R4]    ; A のアドレスをベースとして，B にアクセス
    MOV.L       #3,8[R4]    ; A のアドレスをベースとして，C にアクセス
```

［備考］

- 外部変数もしくは静的変数の定義順を変更した場合は，外部シンボルアドレス情報ファイルを生成し直す必要があります。map オプション以外で 1 回目のコンパイル時に指定したオプションと異なるオプションを指定した場合と，関数内の処理を追加した場合は，動作は保証しません。これらの場合は必ず外部シンボルアドレス情報ファイルを生成し直してください。
- C/C++ ソースをコンパイルする場合のみ適用されます。コンパイル時に output=src を用いて作成，またはアセンブリ言語で記述されたプログラムには適用されません。
- map オプションと smap オプションを同時に指定した場合は，map オプションが有効となります。
- プログラムセクションの次に連続してデータセクションを配置すると，外部変数アクセス最適化が無効になる，あるいは，最適化が十分に機能しない場合があります。
- 最適化を最大限に機能させるためには，連続して複数のセクションを配置させる場合，プログラムセクションを末尾に配置してください。
- 以下に具体例を示します。

- P を 0x100 番地から，C1,C2 を P の直後，C3 を 0x400 番地から配置したとします。
- この場合，P セクションと連続して C1,C2 セクションが配置されているため，C2 の後方に配置してください。C3 セクションは配置関係が連続していないため無関係です。

-smap

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-smap
```

［詳細説明］

- コンパイル対象ファイル内で定義された外部変数もしくは静的変数についてベースアドレスを設定し，アクセスをベースアドレス相対で行うコードを生成します。

［例］

- ＜C ソース＞

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

- ＜出力コード＞

```
_func:
    MOV.L       #_A,R4      ; A のアドレスをベースアドレスに設定
    MOV.L       #1,[R4]
    MOV.L       #2,4[R4]    ; A のアドレスをベースとして，B にアクセス
    MOV.L       #3,8[R4]    ; A のアドレスをベースとして，C にアクセス
```

［備考］

- C/C++ ソースをコンパイルする場合のみ適用されます。コンパイル時に output=src を用いて作成，またはアセンブリ言語で記述されたプログラムには適用されません。
- map オプションと smap オプションを同時に指定した場合は，map オプションが有効となります。

-nomap

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nomap
```

- 省略時解釈

  optimize=0，optimize=1，または optimize=2 オプションを指定した場合は nomap です。

［詳細説明］

- nomap オプションを指定した場合，外部変数アクセス最適化を行いません。

［例］

- ＜C ソース＞

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

- ＜出力コード＞

```
_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

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-approxdiv
```

- 省略時解釈

  浮動小数点定数除算を，定数の逆数の乗算に変換しません。

［詳細説明］

- 浮動小数点定数除算を，定数の逆数の乗算に変換します。
- 変数÷除数 という式があるとき，除数が定数の場合は，変数×（除数の逆数）に変換したコードを生成します。

［備考］

- 本オプションを指定した場合，浮動小数点定数除算の実行速度は向上しますが，演算の精度と演算の順序が変わる場合がありますので注意が必要です。

-enable_register

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-enable_register
```

［詳細説明］

- V2.00 で、本オプションは無効になりました。指定した場合、無視されますが、従来バージョンとの互換性のためエラーにはなりません。

-simple_float_conv

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-simple_float_conv
```

［詳細説明］

- 浮動小数点型の型変換処理の一部を省略します。
- 本オプション選択時は，次の浮動小数点の型変換を行う生成コードが変化します。

  a) 32bit 浮動小数点型から符号無し整数型への変換

  b) 符号無し整数型から 32bit 浮動小数点型への変換

  c) 32bit 浮動小数点型を経由した，整数型から 64bit 浮動小数点型への変換

［例］

＜ a) 32bit 浮動小数点型から符号無し整数型への変換＞
-fpu の指定が有効でない場合は該当しません。

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

オプション非指定時：
 _func1:
         FCMP    #4F000000H,R1
         BLT     L12
         FADD    #0CF800000H,R1
 L12:
         FTOI    R1,R1
         RTS

オプション指定時：
 _func1:
         FTOI    R1,R1
         RTS
```

＜ b) 符号無し整数型から 32bit 浮動小数点型への変換＞
-fpu の指定が有効でない場合は該当しません。

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

オプション非指定時：
 _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

オプション指定時：
 _func2:
         ITOF    R1,R1
         RTS
```

＜ c) 32bit 浮動小数点型を経由した，整数型から 64bit 浮動小数点型への変換＞
-dbl_size=8 の指定が有効でない場合は該当しません。

```
 double func3(long l)
 {
       return (double)(float)l;
 }
オプション非指定時：
_func3:
        ITOF    R1,R1
        BRA     __COM_CONVfd

オプション指定時：
        BRA     __COM_CONV32sd
```

［備考］

- 本オプション指定時，該当する型変換の処理に対するコード性能は向上しますが，変換結果が C,C++ 言語規格と異なる場合がありますので，ご注意ください。
- -optimize=0 のとき c) は無効になります。

-fpu

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-fpu
```

- 省略時解釈

  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。

  CPU に RX200 を選択（*1）した場合は nofpu，それ以外は fpu です。

注　*1）isa オプションまたは環境変数 ISA_RX による選択を指します。

*2) cpu オプションまたは環境変数 CPU_RX による選択を指します。

[詳細説明]

- fpu を指定した場合，FPU 命令を使用したコード生成を行います。

[備考]

- FPU 命令の具体的な内容については，RX コーディング編の「インストラクション」を参照してください。
- CPU として RX200 が選択されている場合，fpu を指定するとエラーになります。

-nofpu

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nofpu
```

- 省略時解釈

  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。

  CPU に RX200 を選択（*1）した場合は nofpu，それ以外は fpu です。

注　*1）isa オプションまたは環境変数 ISA_RX による選択を指します。

*2) cpu オプションまたは環境変数 CPU_RX による選択を指します。

［詳細説明］

- nofpu を指定した場合，FPU 命令を使用しないコード生成を行います。

-alias

<コンパイル・オプション / 最適化オプション>

[指定形式]

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

- 省略時解釈

  alias=noansi です。

[詳細説明]

- ポインタ指示先の型を考慮した最適化を実施するかどうかを選択します。
- alias=ansi を指定した場合，ANSI 規格に基づき，ポインタ指示先の型を考慮した最適化を行います。一般には，alias=noansi を指定した場合よりもオブジェクト性能が向上しますが，alias=ansi と alias=noansi とで実行結果が異なる場合があります。
- alias=noansi を指定した場合は V.1.00 と同様で，ANSI 規格に基づくポインタ指示先の型を考慮した最適化を行いません。

[例]

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

- ＜ alias=noansi 指定時＞

*ps = 2; によって，n の値が書き換わる可能性があるとみなし（A）で n の値を再ロードします。

```
_func:
     MOV.L       #_n,R4
     MOV.L       #1,[R4]     ;  n = 1;
     MOV.W       #2,[R1]     ;  *ps = 2;
     MOV.L       [R4],R5     ;  （A）n を再ロードする
     MOV.L       #_x,R4
     MOV.L       R5,[R4]
     RTS
```

- ＜ alias=ansi 指定時＞

*ps と n は型が異なるため，*ps = 2; では n の値は変化しないと判断し，（B）で，n = 1 で代入に使用した値を再利用します（もし *ps = 2; によって n の値が書き換わる場合，結果は変わります）。

```
_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）n = 1 で代入に使用した値を再利用する
    RTS
```

［備考］

- optimize=0 または optimize=1 が有効な場合に alias オプションを選択すると，alias=ansi の選択は無視され，常に alias=noansi が選択されたものとしてコード生成します。

-float_order

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-float_order
```

- 省略時解釈

  浮動小数点演算式の演算順序変更の最適化を行いません。

[詳細説明]

- V2.00 で、本オプションは無効になりました。指定した場合、無視されますが、従来バージョンとの互換性のためエラーにはなりません。

-ip_optimize

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-ip_optimize
```

［詳細説明］

- 大域最適化を実施します。
  - 手続き間別名解析を利用した最適化
  - 引数・戻り値の定数伝播　など

［例］

1.

- ＜C ソース＞

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

- <ip_optimize の指定がない場合のアセンブリ出力 >

```
; -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
```

- <ip_optimize 指定時のアセンブリ出力 >

```
; -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 ソース＞

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

- <ip_optimize の指定がない場合のアセンブリ出力 >

```
; -optimize=2 -size

__$func:
  ADD R3, R1
  SUB R2, R1
  RTS

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

- <ip_optimize 指定時のアセンブリ出力 >

```
; -optimize=2 -size

__$func:
  MOV.L #00000004H, R1
  RTS

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

［備考］

- merge_files オプションと同時に指定することにより、ファイル間の最適化も行うことができます。

-merge_files

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-merge_files
```

[詳細説明]

- 複数の C/C++ ソースファイルをマージしてコンパイルすることにより、1 つのオブジェクトファイルを出力します。
- output オプションでファイル名を指定した場合は指定した名前のファイルを生成し、指定しなかった場合は先頭に入力したソースファイル名に出力形式に応じた拡張子をつけたファイルを生成します。
- output オプションで出力形式に src または obj を指定した場合は、他の入力したソースファイル名に出力形式に応じた拡張子をつけた空ファイルを生成します。

[例]

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

files.obj にオブジェクトを生成します。また、空ファイル file1.obj、file2.obj、file3.obj を生成します。

[備考]

- コンパイル対象ファイルが 1 つの場合は、本オプションは無効になります。
- output オプションで出力形式に prep を指定した場合、本オプションは無効になります。
- 本オプションと inline オプションを同時に指定した場合、ファイル間インライン展開も行うことができます。
- C++ または EC++ でコンパイルするコンパイル対象ファイルに対しては、本オプションは無効になります。
- プログラムに static 変数、static 関数を含む場合、次の制限があります。
  - デバッガのウォッチウィンドウで、ファイル間で同じ名前の static 変数を表示する場合は、変数名だけでなくファイル名も指定してください。ファイル名がないと変数が特定できず表示内容が不定となります。
  - ファイル間で同じ名前の static 関数があり、関数が属すセクションを rlink でオーバレイ化した場合、デバッガのオーバレイセクションの優先表示機能は使用できません。
  - リンクマップファイル (.map) には、static 変数と static 関数は、元の名前では表示されず、代わりにコンパイラで変換した名前が表示されます。
- ソースファイル間で、同じ変数の宣言内容 (型指定子など) が異なる場合、コンパイル時にエラーとなる場合があります。
- 本オプションを指定して生成したオブジェクト・ファイルをリンクする際にリンク・オプション -delete, -rename, -replace のいずれかを指定した場合，動作は保証しません。

-whole_program

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-whole_program
```

［詳細説明］

- コンパイル対象ファイルがプログラム全体であることを仮定し、コンパイル対象ファイル全てをマージして大域最適化を実施します。

［備考］

- 本オプションを指定した場合、ip_optimize オプションが有効になります。さらに複数のソースファイルを入力した場合は、merge_files オプションが有効になります。
- 本オプションを指定した場合、コンパイラは以下の条件を満たすことを前提にコンパイルを行います。条件を満たさない場合の動作は保証しません。
  - コンパイル対象ファイル内で定義された extern 変数の内容及びアドレスが、コンパイル対象ファイル以外で設定及び参照されることはない
  - コンパイル対象ファイル内からコンパイル対象ファイル以外の関数を呼び出したとしても、呼び出された関数からコンパイル対象ファイル内の関数が呼ばれることはない

［例］

```
[wp.c]
extern void g(void);
int func(void)
{
    static int a = 0;
    a++;      // (1) a に値を書き込む。
    g();      // (2) g() を呼び出す。
    return a; // (3) a を読み出す。
}
```

[whole_program 指定なし]

関数 g() は関数 func() を呼び出す可能性があるので、(2) の実行で変数 a は書き換わるとみなし、(3) では a の値を読み出すコードを生成します。

```
_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
```

[whole_program 指定あり]

関数 g() は関数 func() 呼び出さないとみなすため、(2) の実行で変数 a は不変と判断します。これを利用し、(3) では a の値を読み出さず、代わりに (1) で a に書き込んだ値を使用するコードを生成します。

```
_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
```

マイコンオプション

<コンパイル・オプション / マイコンオプション>

マイコンオプションには，次のものがあります。

- -isa
- -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

-isa

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-isa={ rxv1 | rxv2 }
```

- 省略時解釈

  環境変数 ISA_RX の内容に従います。

[詳細説明]

- 生成する命令コードの命令セットアーキテクチャ (RXv1 または RXv2) を指定します。
- isa=rxv1 を指定した場合，RXv1 を選択します。
- isa=rxv2 を指定した場合，RXv2 を選択します。

[備考]

- -nofpu と -fpu のいずれの選択もない場合に -isa オプションを指定すると、-fpu を自動的に選択します。
- -isa オプションを省略した場合、-cpu オプション、環境変数 CPU_RX および 環境変数 ISA_RX のいずれもない場合は、エラーとなります。
- -cpu オプションと同時に指定することはできません。

-cpu

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  環境変数 CPU_RX に従います。

[詳細説明]

- マイコン種別を指定します。
- cpu=rx600 を指定した場合，RX600 向けの命令コードを生成します。
- cpu=rx200 を指定した場合，RX200 向けの命令コードを生成します。

[備考]

- 本オプションは、従来機能と互換性を保つためのものです。
- 今後、RX ファミリの製品展開で追加される品種については、命令セットアーキテクチャなどの選択は、-cpu オプションではなく、-isa オプションでの対応となります。新規にアプリケーション開発する際は、-isa オプションを指定してください。
- -cpu オプションは、次の記述により -isa オプションと -fpu,-nofpu オプションで置き換えることが可能です。*1
  - -cpu=rx600 → -isa=rxv1 -fpu
  - -cpu=rx200 → -isa=rxv1 -nofpu
- -cpu=rx200 を指定すると，-nofpu が自動的に選択されます。
- -cpu=rx200 と fpu は同時に指定することはできません。
- -nofpu と fpu のいずれの選択もない場合に -cpu=rx600 を指定すると，-fpu を有効にします。
- - cpu オプションを省略し、-isa オプション、環境変数 CPU_RX および 環境変数 ISA_RX のいずれも指定していない場合はエラーとなります。
- -isa オプションと同時に指定することはできません。

【注意】

- *1) ソースプログラムにプリデファインドマクロ __RX200、__RX600 の記述がある場合を除きます。
-

-endian

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  endian=little です。

[詳細説明]

- endian=big を指定した場合，データのバイト並びが big endian になります。
- endian=little を指定した場合，データのバイト並びが little endian になります。
- #pragma endian 拡張子でも指定できます。オプションと #pragma 拡張子の両方が指定された場合には，
  #pragma 拡張子の指定を優先します。

[例]

-

[備考]

-

-round

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  round=nearest です。

[詳細説明]

- 浮動小数点定数演算の丸め方式を選択します。
- round=zero を指定した場合，round to zero で丸めます。
- round=nearest を指定した場合，round to nearest で丸めます。

[例]

-

[備考]

- 本オプションでは，実行時の浮動小数点演算における丸め方式を変更することはできません。
- 本オプションのデフォルトの選択は，fpu, nofpu オプション選択の影響を受けません。

-denormalize

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  denormalize=off です。

[詳細説明]

- 浮動小数点定数に非正規化数を記述した場合の扱いを指定します。
- denormalize=off を指定した場合，非正規化数を 0 として扱います。
- denormalize=on を指定した場合，非正規化数を非正規化数として扱います。

[例]

-

[備考]

- 本オプションでは，実行時の浮動小数点演算における非正規化数の扱いを変更することはできません。
- 本オプションは，fpu, nofpu オプション選択で自動的に有効になることはありません。

-dbl_size

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈

  dbl_size=4 です。

［詳細説明］

- double 型，および long double 型の精度を指定します。
- dbl_size=4 を指定した場合，単精度浮動小数点型（4 バイト）として扱います。
- dbl_size=8 を指定した場合，倍精度浮動小数点型（8 バイト）として扱います。

［例］

-

［備考］

- dbl_size=4 を選択した場合，標準関数のうち mathf.h と math.h とで同じ仕様の関数（例 : sqrtf と sqrt など）を一体化して標準ライブラリが構築されます。このため dbl_size=4 選択時は，例えば mathf.h ヘッダの関数である sqrtf を呼び出すところを，RX のシミュレータやエミュレータでトレース（ステップ実行）すると，sqrtf ではなく同じ仕様を持つ math.h ヘッダの関数 sqrt が呼び出されたように見えることがあります。

-int_to_short

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-int_to_short
```

- 省略時解釈

  ソースファイル内の int を short に，unsigned int を unsigned short に置換せずコンパイルを行います。

[詳細説明]

- ソースファイル内の int を short に，unsigned int を unsigned short に置換してコンパイルを行います。

[例]

-

[備考]

- limits.h の INT_MAX，INT_MIN，および UINT_MAX は本オプションの変換対象外となります。
- C++ および EC++ コンパイル時は，本オプションは無効になります。C++，EC++ プログラム内から C プログラムを参照する可能性がある外部名に対して W0523041 を出力します。
- C 標準ヘッダをインクルードしたファイルを int_to_short オプションを指定して C++ または EC++ コンパイルした場合も，W0523041 が出力されることがあります。この場合は動作には問題ありませんので無視してください。
- C と C++（EC++）との間で共通にアクセスするデータは，int 型ではなく long 型または short 型で宣言してください。

-signed_char

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-signed_char
```

- 省略時解釈

  char 型を unsigned char 型として扱います。

[詳細説明]

- signed char 型として扱います。

[備考]

- char 型のビットフィールドメンバは，本オプションの制御対象外です。signed_bitfield および unsigned_bitfield オプションで制御してください。

-unsigned_char

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-unsigned_char
```

- 省略時解釈

  char 型を unsigned char 型として扱います。

［詳細説明］

- unsigned_char を指定した場合，unsigned char 型として扱います。

［備考］

- char 型のビットフィールドメンバは，本オプションの制御対象外です。signed_bitfield および unsigned_bitfield オプションで制御してください。

-signed_bitfield

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-signed_bitfield
```

- 省略時解釈

  符号なし型として扱います。

[詳細説明]

- 符号付き型として扱います。

-unsigned_bitfield

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-unsigned_bitfield
```

- 省略時解釈

  符号なし型として扱います。

[詳細説明]

- 符号なし型として扱います。

-auto_enum

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-auto_enum
```

- 省略時解釈

  列挙型サイズを signed long 型として処理します。

［詳細説明］

- enum 宣言した列挙型のデータを，列挙値が収まる最小型として処理します。
- 列挙型のとりうる値と型の関係を以下に示します。

表 A—6　列挙型のとりうる値と型の関係

列挙子		選択される型
最小値	最大値	
-128	127	signed char
0	255	unsigned char
-32768	32767	signed short
0	65535	unsigned short
上記以外		signed long

-bit_order

<コンパイル・オプション / マイコンオプション>

［指定形式］

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

- 省略時解釈

  bit_order=right です。

［詳細説明］

- ビットフィールドのメンバの並び順を指定します。
- bit_order=left を指定した場合は上位ビットからメンバを割り付けます。
- bit_order=right を指定した場合は下位ビットからメンバを割り付けます。
- #pragma bit_order 拡張子でも指定できます。オプションと #pragma の両方が指定された場合には，拡張子の指定を優先します。

-pack

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-pack
```

- 省略時解釈

  構造体，クラスのアライメント数は，メンバの最大のアライメント数と同じになります。

[詳細説明]

- 構造体メンバ，クラスメンバのアライメント数を指定します。
- 構造体メンバのアライメント数は，#pragma pack 拡張子でも指定できます。オプションと #pragma の両方が指定された場合には，#pragma 拡張子の指定を優先します。構造体，クラスのアライメント数は，メンバの最大のアライメント数と同じになります。

[例]

-

[備考]

- 本オプション指定時の構造体メンバのアライメント数を以下に示します。

表 A—7　pack オプション指定時の構造体メンバ，クラスメンバのアライメント数

メンバの型	pack	指定なし
(signed) char	1	1
(unsigned) short	1	2
(unsigned) int 注, (unsigned) long, (unsigned) long long, 浮動小数点型 , ポインタ型	1	4

注　int_to_short オプションを指定した場合は，short と同じになります。

-unpack

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-unpack
```

- 省略時解釈

  構造体，クラスのアライメント数は，メンバの最大のアライメント数と同じになります。

[詳細説明]

- 構造体メンバ，クラスメンバのアライメント数を指定します。
- 構造体，クラスのアライメント数は，メンバの最大のアライメント数と同じになります。

[例]

-

[備考]

- 本オプション指定時の構造体メンバのアライメント数を以下に示します。

表 A—8　unpack オプション指定時の構造体メンバ，クラスメンバのアライメント数

メンバの型	unpack	指定なし
(signed) char	1	1
(unsigned) short	2	2
(unsigned) int 注, (unsigned) long, (unsigned) long long, 浮動小数点型 , ポインタ型	4	4

注　int_to_short オプションを指定した場合は，short と同じになります。

-exception

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-exception
```

- 省略時解釈

  C++ 例外処理機能（try, catch, throw）を無効にします。

[詳細説明]

- C++ 例外処理機能（try, catch, throw）を有効にします。
- コード性能が低下する可能性があります。

[備考]

- ファイル間で例外処理機能を有効にするには以下を行ってください。
  - rtti=on を指定する。
  - 最適化リンケージエディタで noprelink オプションを指定しない。
- exception オプションは C++ コンパイル時にのみ指定できます。lang=cpp の指定がなく，かつ入力ファイルの拡張子が .c または .p の場合，exception オプションを指定しても無視されます。

-noexception

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-noexception
```

- 省略時解釈

  C++ 例外処理機能（try, catch, throw）を無効にします。

[詳細説明]

- C++ 例外処理機能（try, catch, throw）を無効にします。

[備考]

- ファイル間で例外処理機能を有効にするには以下を行ってください。
  - rtti=on を指定する。
  - 最適化リンケージエディタで noprelink オプションを指定しない。
- noexception オプションは C++ コンパイル時にのみ指定できます。lang=cpp の指定がなく，かつ入力ファイルの拡張子が .c または .p の場合，noexception オプションは指定できません。指定するとエラーとなります。

-rtti

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  rtti=off です。

[詳細説明]

- 実行時型情報の有効 / 無効を指定します。
- rtti=on を指定した場合，dynamic_cast，typeid を有効にします。
- rtti=off を指定した場合，dynamic_cast，typeid を無効にします。

[備考]

- 本オプションを指定して作成したリロケータブルファイル（.obj）をライブラリに登録したり，最適化リンケージエディタでリロケータブル形式（.rel）で出力しないでください。シンボルの二重定義エラーや未定義エラーになることがあります。
- rtti=on は，C++ コンパイル時にのみ指定できます。lang=cpp の指定がなく，かつ入力ファイルの拡張子が .c または .p の場合，rtti=on を指定しても無視されます。

-fint_register

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈

  fint_register=0 です。

［詳細説明］

- 高速割り込み関数（#pragma interrupt で割り込み仕様に高速割り込み指定（fint）のある関数）でのみ使用する汎用レジスタを指定します。高速割り込み関数以外では，指定されたレジスタは使用しません。本オプションで指定した汎用レジスタは，高速割り込み関数内では退避回復なしで使用できるため，高速割り込み関数の高速化が見込めます。反面，他の関数で使用可能な汎用レジスタが減るため，プログラム全体のレジスタ割付効率は低下します。
- オプションとレジスタの関係を以下に示します。

表 A—9　オプションとレジスタの関係

オプション	高速割り込み専用レジスタ
fint_register=0	なし
fint_register=1	R13
fint_register=2	R12, R13
fint_register=3	R11, R12, R13
fint_register=4	R10, R11, R12, R13

［備考］

- 高速割り込み関数以外で，本オプションで指定したレジスタを使用した場合の動作は保証しません。本オプションの指定の対象となるレジスタが，base オプションで指定されていた場合，エラーとなります。

-branch

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  branch=24 です。

[詳細説明]

- 分岐幅を指定します。
- branch =16 を指定した場合，分岐幅が 16bit 以内であるとしてコンパイルします。
- branch =24 を指定した場合，分岐幅が 24bit 以内であるとしてコンパイルします。
- branch =32 を指定した場合，分岐幅を指定しません。

-base

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-base = {  rom=< レジスタ >
         | ram=< レジスタ >
         | < アドレス値 > = < レジスタ >}
                          < レジスタ >:= {R8 ～ R13}
```

[詳細説明]

- プログラム全体で，ベースアドレスとして固定で使用する汎用レジスタを指定します。
- base=rom=< レジスタ A> を指定した場合は，const 変数のアクセスを指定したレジスタ A 相対で行うことを試みます。ただし，定数領域セクション全体の大きさが 64KB ～ 256KB*1 以内でなければなりません。
- base=ram=< レジスタ B> を指定した場合は，初期化変数および未初期化変数のアクセスは指定したレジスタ B 相対で行います。ただし，RAM データ全体の大きさが 64KB ～ 256KB*1 以内でなければなりません。
- < アドレス値 >=< レジスタ C> を指定した場合は，コンパイル時に割り付けアドレスが確定している領域のうち，アドレス値から 64KB ～ 256KB*1 以内の領域のアクセスを，指定したレジスタ C 相対で行います。

注　*1　この値は，アクセスする変数のサイズにより，64KB から 256KB の間で変化します。

[備考]

- 異なる領域に対して同じレジスタを指定することはできません。
- レジスタはそれぞれの領域で 1 個だけ指定可能です。fint_register オプションで指定したレジスタを本オプションで指定した場合，エラーとなります。
- pid オプション選択時は，base=rom=< レジスタ > を選択できません。選択すると，警告として W0523039 メッセージを表示して base=rom=< レジスタ > の選択を無効とします。

-patch

<コンパイル・オプション / マイコンオプション>

[指定形式]

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

[詳細説明]

- CPU の品種ごとに特有の問題を回避します。
  - -patch=rx610 を指定すると，RX610 グループで問題となる，MVTIPL 命令を生成コードに使用しません。
  - -patch=rx610 を指定しなければ，組み込み関数 set_ipl の呼び出しに対する生成コードは，MVTIPL 命令を含んだものとなります。

-pic

<コンパイル・オプション / マイコンオプション>

[指定形式]

```
-pic
```

- 省略時解釈

  プログラムセクションを PIC（位置独立コード）としてコード生成しません。

[詳細説明]

- プログラムセクションを PIC（位置独立コード）としてコード生成します。
- PIC においては，関数呼び出しはすべて BSR または BRA 命令を用いて行い，また関数のアドレスを取得するときは PC からの相対アドレスを用いるようにします。これによって，PIC はリンク後に任意のアドレスに配置することができます。

[例]

- 関数呼び出し （ただし，branch=32 の場合）

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

[-pic なし ]
_func:
            MOV.L       #_sub,R14
            JMP         R14

[-pic あり ]
_func:
            MOV.L       #_sub-L11,R14
L11:
            BRA         R14
```

- 関数アドレスの取得

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

[-pic なし ]
_func2:
            MOV.L       #_f_ptr,R4
            MOV.L       #_func1,[R4]
            RTS

[-pic あり ]
_func2:
            MOV.L       #_f_ptr,R4
L11:
            MVFC        PC,R14
            ADD         #_func1-L11,R14
            MOV.L       R14,[R4]
            RTS
```

［備考］

- C++ または EC++ コンパイル時は，pic オプションを選択できません。選択すると，警告として W0523039 メッセージを表示して pic の選択を無効とします。
- PIC である関数のアドレスを，静的初期化の初期化式に使用しないでください。エラー E0523026 となります。
- PIC アドレスを静的初期化に用いる例 :

```
void pic_func1(void), pic_func2(int), pic_func3(int);     /* PIC になる       */
void (*fptr1_for_pic) = pic_func1;      /* PIC アドレスを静的初期化で使用： エラー */
struct PIC_funcs{ int code; void (*fptr)(int); };
struct PIC_funcs pic_funcs[] = {
    { 2, pic_func2 },                   /* PIC アドレスを静的初期化で使用： エラー */
    { 3, pic_func3 },                   /* PIC アドレスを静的初期化で使用： エラー */
};
```

- PIC 機能を利用するアプリケーションプログラムのスタートアップを作成する際は，「スタートアップ」ではなく，RX コーディング編の「アプリケーションのスタートアップ」を参照ください。
- PIC 機能については，RX コーディング編の「PIC/PID 機能の利用」の項目も参照してください。

-pid

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈

  定数領域セクション C, C_2 および C_1，リテラルセクション L と switch 文分岐テーブルセクション W, W_2 および W_1 を PID（位置独立データ）としません。

［詳細説明］

- 定数領域セクション C, C_2 および C_1，リテラルセクション L と switch 文分岐テーブルセクション W, W_2 および W_1 を PID（位置独立データ）とします。
- PID のアクセスは，すべて PID レジスタからの相対アドレスで行います。これにより，PID はリンク後に任意のアドレスに配置することができます。
- PID 機能を実現するためには，汎用レジスタを 1 本消費します。
- ＜PID レジスタ＞
  - 下記の表の規則に基づき，fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお，fint_register の指定がない場合は R13 を選択します。

表 A—10　fint_register オプションと PID レジスタの関係

fint_register オプション	PID レジスタ
fint_register 指定なし	R13
fint_register=0	
fint_register=1	R12
fint_register=2	R11
fint_register=3	R10
fint_register=4	R9

  - PID レジスタは，PID のアクセスに使用する用途以外には使用されません。
- ＜パラメータ＞
  - パラメータは，PID レジスタから定数領域セクションをアクセスする際の，オフセットの最大幅のビット数を 16 または 32 で選択します。
  - オフセット幅を省略して pid オプションを選択した場合の解釈は，pid=16 です。pid=16 では，PID レジスタがアクセスできる定数領域セクションのサイズが 64KB ～ 256KB（アクセス幅により変動する）に制限されま

す。pid=32 では，PID レジスタがアクセスできる定数領域セクションのサイズに制限はありませんが，PID をアクセスするコードのサイズが増大します。

- なお，pid=32 と 外部シンボル割り付け情報が有効な map オプションを同時指定した場合は，割り付け情報により，PID レジスタによるアクセスが可能な場合は pid=16 と同じコードを生成します。

［例］

- const 修飾した外部参照シンボルをアクセス

```
extern const int pid;
int work;
void func1()
{
        work = pid;
}
[-pid なし ]
_func1:
            MOV.L       #_pid,R4
            MOV.L       [R4],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
[-pid=16 あり ] （※ただし，PID レジスタが R13 の場合）
_func1:
            MOV.L       _pid-__PID_TOP:16[R13],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
[-pid=32 あり ] （※ただし，PID レジスタが R13 の場合）
_func1:
            ADD         #(_pid-__PID_TOP),R13,R6
            MOV.L       [R6],R5
            MOV.L       #_work,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
```

- const 修飾した外部定義シンボルのアドレスを取得

```
extern const int pid = 1000;
const int *ptr;
void func2()
{
        ptr = &pid;
}
[-pid なし ]
_func2:
            MOV.L       #_ptr,R4
            MOV.L       #_pid,[R4]
            RTS
[-pid あり ] （※ただし，PID レジスタが R13 の場合）
_func2:
            ADD         #(_pid-__PID_TOP),R13,R5
            MOV.L       #_ptr,R4
            MOV.L       R5,[R4]
            RTS
            .glb        __PID_TOP
```

［備考］

- PID である領域のアドレスを，静的初期化の初期化式に使用しないでください。エラー E0523027 となります。
- PID アドレスを静的初期化に用いる例 :

```
extern const int pid_data1;            /* PID になる                          */
const int *ptr1_for_pid = &pid_data1;  /* PID のアドレスで静的初期化： エラー */
const int pid_data4[] = {1,2,3,4};     /* PID になる                          */
const int *ptr2_for_pid = pid_data4;   /* PID のアドレスで静的初期化： エラー */
```

- PID 機能を利用するアプリケーションプログラムのスタートアップを作成する際は，「スタートアップ」ではなく，「アプリケーションのスタートアップ」を参照ください。
- pid オプション選択時は，同一の外部変数は，必ずファイル間で const 修飾を統一してください。これは，pid オプションは const 修飾のある変数を PID にするためです。もし const 修飾の統一が不明な外部変数がある場合は，pid オプション（PID 機能）は使用しないでください。
- pid オブション選択時に，ファイル指定のある map オプションを有効にした場合，ファイル間で同じ外部変数に対して const 修飾が統一されていない外部参照変数に対して警告 W0530809 を表示することがあります。表示された変数は PID として扱います。
- C++ または EC++ コンパイル時は，pid オプションを選択できません。選択すると，警告 W0523039 を表示して pid の選択を無効とします。
- pid オプション選択時は，base=rom=< レジスタ > を選択できません。選択すると，警告 W0523039 を表示して base=rom=< レジスタ > の選択を無効とします。
- pid オプションで選択された PID レジスタが，base オプションでも選択された場合は警告 W0511113 になります。

- pid オプションと nouse_pid_register オプションを同時に選択するとエラー E0511150 になります。
- アプリケーションおよび PID 機能の内容については，「PIC/PID 機能の利用」を参照してください。

-nouse_pid_register

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-nouse_pid_register
```

- 省略時解釈

  なし

［詳細説明］

- PID レジスタを使用せずにコード生成を行います。
- PID レジスタは、fint_register オプションの指定内容に基づき、pid オプションと同じ規則で選択します。PID レジスタとして選択されるレジスタは、pid オプションの項目を参照してください。
- PID 機能が有効なアプリケーションプログラムから呼び出されるマスタプログラムは，本オプションでアセンブルする必要があります。このとき，アプリケーションに fint_register オプションの選択がある場合は，マスタプログラムにも同じパラメータの fint_register を選択してください。

［備考］

- nouse_pid_register オプションと pid オプションを同時に選択するとエラー E0511150 になります。
- PID レジスタとして選択されるレジスタが，base オプションでも選択された場合は警告 W0511149 になります。
- PID 機能の詳細については，「PIC/PID 機能の利用」の項目を参照してください。

-save_acc

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-save_acc
```

- 省略時解釈

  割り込み関数に対して，アキュムレータ（ACC, ACC0, ACC1）の退避・回復コードを生成しません。

［詳細説明］

- 割り込み関数に対して，アキュムレータ（ACC, ACC0, ACC1）の退避・回復コードを生成します。
- ACC に対する退避・回復コードは、ISA(*1) に RXv1 を選択するか、CPU でマイコン種別を選択 (*2) した場合に生成します。
- ACC0 および ACC1 に対する退避・回復コードは、ISA(*1) に RXv2 を選択した場合に生成します。

注　*1）isa オプションまたは環境変数 ISA_RX による選択を指します。

*2) cpu オプションまたは環境変数 CPU_RX による選択を指します。

［備考］

- 生成される退避・回復コードは，#pragma interrupt に acc を選択したときに生成されるコードと同じものです。実際の退避・回復コードは，「#pragma キーワード」の #pragma interrupt (acc，noacc) の項目を参照ください。
- 本オプション指定時は、割り込み時でもアキュムレータの値が保持されるため、C/C++ の演算式に対して MACW 命令などのアキュムレータを用いる命令を生成することがあります。

アセンブル・リンクオプション

＜コンパイル・オプション / アセンブル・リンクオプション＞

アセンブル，リンクオプションには，次のものがあります。

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

-asmcmd

＜コンパイル・オプション / アセンブル・リンクオプション＞

［指定形式］

```
-asmcmd=< ファイル名 >
```

- 省略時解釈

  なし

［詳細説明］

- asrx に渡すアセンブラオプションをサブコマンドファイルにより指定します。

［例］

```
ccrx -isa=rxv1 -asmcmd=file.sub sample.c
```

と記述した場合，以下の 2 行のコマンド記述と同じ意味になります。

```
ccrx -isa=rxv1 -output=src sample.c
asrx -isa=rxv1 -subcommand=file.sub sample.src
```

［備考］

- 本オプションを複数回指定した場合，指定したすべてのサブコマンドファイルが有効となります。

-lnkcmd

<コンパイル・オプション / アセンブル・リンクオプション>

[指定形式]

```
-lnkcmd=< ファイル名 >
```

- 省略時解釈

  なし

[詳細説明]

- rlink に渡すリンクオプションをサブコマンドファイルにより指定します。

[例]

```
ccrx -isa=rxv1 -output=abs=tp.abs –lnkcmd=file.sub tp1.c tp2.c
```

と記述した場合，以下の 3 行のコマンド記述と同じ意味になります。

```
ccrx -isa=rxv1 -output=src tp1.c tp2.c
asrx -isa=rxv1 tp1.src tp2.src
rlink -subcommand=file.sub -form=abs -output=tp tp1.obj tp2.obj
```

[備考]

- 本オプションを複数回指定した場合，指定したすべてのサブコマンドファイルが有効となります。

-asmopt

<コンパイル・オプション / アセンブル・リンクオプション>

[指定形式]

```
-asmopt=["] <アセンブラオプション> ["]
```

- 省略時解釈

  なし

[詳細説明]

- asrx に渡すアセンブラオプションを文字列により指定します。
- 空白をパラメータに含むオプションを指定する場合は，ダブルクォーテーション（"）で囲んで指定します。

[例]

```
ccrx -isa=rxv1 -asmopt="-chkpm" sample.c
```

と記述した場合，以下の 2 行のコマンド記述と同じ意味になります。

```
ccrx -isa=rxv1 -output=src sample.c
asrx -isa=rxv1 -chkpm sample.src
```

[備考]

- 本オプションを複数回指定した場合，指定したすべてのアセンブラオプションが有効となります。

-lnkopt

<コンパイル・オプション / アセンブル・リンクオプション>

[指定形式]

```
-lnkopt=["] <リンクオプション> ["]
```

- 省略時解釈

  なし

[詳細説明]

- rlink に渡すリンクオプションを文字列により指定します。
- 空白をパラメータに含むオプションを指定する場合は，ダブルクォーテーション（"）で囲んで指定します。

[例]

```
ccrx -isa=rxv1 -output=abs=tp.abs -lnkopt="-start=P,C,D/100,B/8000" tp1.c tp2.c
```

と記述した場合，以下の 3 行のコマンド記述と同じ意味になります。

```
ccrx -isa=rxv1 -output=src tp1.c tp2.c
asrx -isa=rxv1 tp1.src tp2.src
rlink -start=P,C,D/100,B/8000 -form=abs -output=tp tp1.obj tp2.obj
```

[備考]

- 本オプションを複数回指定した場合，指定したすべてのリンクオプションが有効となります。

その他オプション

<コンパイル・オプション / その他オプション>

その他オプションには，次のものがあります。

-logo

<コンパイル・オプション / その他オプション>

[指定形式]

```
-logo
```

- 省略時解釈

  コピーライト表示が出力されます。

[詳細説明]

- コピーライト表示が出力されます。

[例]

-

[備考]

-

-nologo

<コンパイル・オプション / その他オプション>

[指定形式]

```
-nologo
```

- 省略時解釈

  コピーライト表示が出力されます。

[詳細説明]

- nologo オプション指定時は，コピーライトの表示の出力が抑止されます。

-euc

<コンパイル・オプション / その他オプション>

[指定形式]

```
-euc
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を SJIS コードで扱います。

[詳細説明]

- 文字列，文字定数およびコメント内の文字を euc コードで扱います。

-sjis

<コンパイル・オプション / その他オプション>

[指定形式]

```
-sjis
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を SJIS コードで扱います。

[詳細説明]

- 文字列，文字定数およびコメント内の文字を SJIS コードで扱います。

-latin1

<コンパイル・オプション / その他オプション>

［指定形式］

```
-latin1
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を latin1 コードで扱います。

-utf8

<コンパイル・オプション / その他オプション>

[指定形式]

```
-utf8
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を SJIS コードで扱います。

[詳細説明]

- 文字列，文字定数およびコメント内の文字を utf8 コードで扱います。

[備考]

- utf8 オプションは，lang=c99 オプション指定時のみ有効です。

-big5

<コンパイル・オプション / その他オプション>

[指定形式]

```
-big5
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を BIG5 コードで扱います。

[詳細説明]

- 文字列，文字定数およびコメント内の文字を big5 コードで扱います。

[備考]

- big5 を指定した場合は、outcode にも同じ文字コードを指定しなければなりません。

-gb2312

<コンパイル・オプション / その他オプション>

[指定形式]

```
-gb2312
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を GB2312 コードで扱います。

[詳細説明]

- 文字列，文字定数およびコメント内の文字を gb2312 コードで扱います。

[備考]

- gb2312 を指定した場合は、outcode にも同じ文字コードを指定しなければなりません。

-outcode

＜コンパイル・オプション / その他オプション＞

［指定形式］

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

- 省略時解釈

  outcode=sjis です。

［詳細説明］

- 文字列，文字定数内の文字を指定した文字コードで出力します。
- オプションと文字コードの関係を以下に示します。

表 A—11　オプションと文字コードの関係（outcode）

オプション	文字コード
euc	EUC コード
sjis	SJIS コード
latin1	ISO-Latin1 コード
utf8	UTF-8 コード
big5	Big5 コード
gb2312	GB2312 コード

［備考］

- outcode=utf8 の指定は、lang=c99 オプション指定時のみ有効です。
- outcode=big5 または outcode=gb2312 を指定する場合は、それぞれ big5 または gb2312 オプションもあわせて指定しなければなりません。

-subcommand

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-subcommand=< サブコマンドファイル名 >
```

- 省略時解釈

  なし

［詳細説明］

- subcommand オプション指定時は，コンパイラ起動時のコンパイラオプションをサブコマンドファイルで指定します。
- サブコマンドファイル中の書式は，コマンドラインの書式と同一です。

［例］

-

［備考］

- 本オプションを複数回指定した場合，指定したすべてのサブコマンドファイルが有効となります。

(2) アセンブル・オプション

分類	オプション	説明
ソースオプション	-include	インクルード・ファイルの取り込み先フォルダを指定します。
	-define	マクロ定義を指定します。
	-chkpm	特権命令のチェックします。
	-chkfpu	浮動小数点演算命令のチェックします。
	-chkdsp	DSP 機能命令のチェックします。
オブジェクトオプション	-output	リロケータブル・ファイル名を指定します。
	-debug	オブジェクト・ファイルにデバッグ情報を出力します。
	-nodebug	オブジェクト・ファイルにデバッグ情報を出力しません。
	-goptimize	モジュール間最適化用付加情報を出力します。
	-fpu	FPU 命令が有効であるリロケータブルファイルを生成します。
	-nofpu	FPU 命令が有効ではないリロケータブルファイルを生成します
リストオプション	-listfile	アセンブル・リスト・ファイル出力します。
	-nolistfile	アセンブル・リスト・ファイル出力しません。
	-show	ソース・リスト・ファイルの内容を指定します。
マイコンオプション	-isa	命令セット・アーキテクチャを選択します。
	-cpu	マイコン種別を選択します。
	-endian	エンディアン選択します。
	-fint_register	高速割り込み関数でのみ使用する汎用レジスタを選択します。
	-base	ROM,RAM 用ベースレジスタ選択します。
	-patch	CPU タイプ特有の問題を回避するかどうかを選択します。
	-pic	PIC 機能を有効にします。
	-pid	PID 機能を有効にします。
	-nouse_pid_register	PID レジスタをコード生成に使用しません。
その他オプション	-logo	コピーライトを出力します。
	-nologo	コピーライトを出力しません。
	-subcommand	コマンドオプションを取り込むファイルを指定します。
	-euc	入力プログラムの文字コードを EUC コードと解釈します。
	-sjis	入力プログラムの文字コードを SJIS コードと解釈します。
	-latin1	入力プログラムの文字コードを ISO-Latin1 コードと解釈します。
	-big5	入力プログラムの文字コードを BIG5 コードと解釈します。
	-gb2312	入力プログラムの文字コードを GB2312 コードと解釈します。

ソースオプション

＜アセンブル・オプション / ソースオプション＞

ソースオプションには，次のものがあります。

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

-include

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-include = <パス名>[,…]
```

- 省略時解釈

  インクルードファイルの検索は，カレントフォルダ，環境変数 INC_RXA 指定フォルダの順序で行います。

［詳細説明］

- インクルードファイルの存在するパス名を指定します。
- パス名が複数ある場合にはカンマ（,）で区切って指定することができます。
- インクルードファイルの検索は，カレントフォルダ，include オプション指定フォルダ，環境変数 INC_RXA 指定フォルダの順序で行います。

［例］

- フォルダ c:￥usr￥inc と c:￥usr￥rxc をインクルードファイルパスとして検索します。

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

-define

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-define = <sub>[,…]
          <sub> : < マクロ名 > = < 文字列 >
```

- 省略時解釈

  なし

［詳細説明］

- マクロ名を対応する文字列に置き換えます。

  （ソースファイル先頭で .DEFINE 指示命令を記述するのと同じです）

［備考］

- define オプションと .DEFINE が同時指定された場合，.DEFINE を優先します。

-chkpm

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkpm
```

- 省略時解釈

  なし

［詳細説明］

- 特権命令を記述するとウォーニング A1011 を通知します。

［備考］

- 特権命令の詳細な説明については，RX コーディング編の「インストラクション」を参照してください。

-chkfpu

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkfpu
```

- 省略時解釈

  なし

［詳細説明］

- 本オプションを指定した場合，浮動小数点演算命令を記述するとウォーニング A1012 を通知します。

［備考］

- 浮動小数点演算命令の詳細な説明については，RX コーディング編の「インストラクション」を参照してください。

-chkdsp

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkdsp
```

- 省略時解釈

  なし

［詳細説明］

- 本オプションを指定した場合，DSP 機能命令を記述するとウォーニング A1013 を通知します。

［備考］

- DSP 機能命令の詳細な説明については，RX コーディング編の「インストラクション」を参照してください。

オブジェクトオプション

＜アセンブル・オプション / オブジェクトオプション＞

オブジェクトオプションには，次のものがあります。

- -output
- -debug
- -nodebug
- -goptimize
- -fpu
- -nofpu

-output

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-output = < 出力ファイル名 >
```

- 省略時解釈

  ソースファイルと同じファイル名で拡張子が「obj」のリロケータブルファイルを出力します。

［詳細説明］

- 出力するリロケータブルファイル名は，出力ファイル名に拡張子がない場合，出力ファイル名に拡張子「.obj」を付加した文字列となり，拡張子がある場合，出力ファイル名の拡張子を「.obj」で置き換えた文字列となります。

-debug

<アセンブル・オプション / オブジェクトオプション>

[指定形式]

```
-debug
```

- 省略時解釈

  リロケータブルファイル内にデバッグ情報を出力しません。

[詳細説明]

- リロケータブルファイル内にデバッグ情報を出力します。

-nodebug

<アセンブル・オプション / オブジェクトオプション>

[指定形式]

```
-nodebug
```

- 省略時解釈

  リロケータブルファイル内にデバッグ情報を出力しません。

[詳細説明]

- リロケータブルファイル内にデバッグ情報を出力しません。

-goptimize

<アセンブル・オプション / オブジェクトオプション>

[指定形式]

```
-goptimize
```

- 省略時解釈

  モジュール間最適化用付加情報を出力しません。

[詳細説明]

- モジュール間最適化用付加情報を出力します。
- 本オプションを指定したファイルは，リンク時にモジュール間最適化の対象になります。

-fpu

<アセンブル・オプション / オブジェクトオプション>

[指定形式]

```
-fpu
```

- 省略時解釈

  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。

  CPU に RX200 を選択（*1）した場合は nofpu，それ以外は fpu です。

注　*1）isa オプションまたは環境変数 ISA_RX による選択を指します。

*2) cpu オプションまたは環境変数 CPU_RX による選択を指します。

[詳細説明]

- FPU 命令が有効であるリロケータブルファイルを生成します。]

[備考]

- CPU に RX200 を選択した場合，fpu を指定するとエラーになります。
- FPU 命令の具体的な内容については，RX コーディング編の「インストラクション」を参照してください。

-nofpu

<アセンブル・オプション / オブジェクトオプション>

[指定形式]

```
-nofpu
```

- 省略時解釈

  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。

  CPU に RX200 を選択（*1）した場合は nofpu，それ以外は fpu です。

注　*1）isa オプションまたは環境変数 ISA_RX による選択を指します。

*2) cpu オプションまたは環境変数 CPU_RX による選択を指します。

[詳細説明]

- FPU 命令が有効ではないリロケータブルファイルを生成します。

[備考]

- FPU 命令の具体的な内容については，RX コーディング編の「インストラクション」を参照してください。
- 本オプションを指定した場合、浮動小数点演算命令と、制御レジスタ FPSW の記述をエラーとします。

リストオプション

＜アセンブル・オプション / リストオプション＞

リストオプションには，次のものがあります。

- -listfile
- -nolistfile
- -show

-listfile

＜アセンブル・オプション / リストオプション＞

［指定形式］

```
-listfile[=< ファイル名 >]
```

- 省略時解釈

  アセンブルリストファイルは出力しません。

［詳細説明］

- アセンブルリストファイルを出力します。＜ファイル名＞を指定することもできます。
- ＜ファイル名＞は，「ファイル名の付け方」に従って指定できます。
- listfile オプションで＜ファイル名＞を指定しない場合には，ソースファイルと同じファイル名で，拡張子が「lst」のアセンブルリストファイルが作成されます。

-nolistfile

＜アセンブル・オプション / リストオプション＞

［指定形式］

```
-nolistfile
```

- 省略時解釈

  アセンブルリストファイルは出力しません。

［詳細説明］

- アセンブルリストファイルは出力しません。

-show

＜アセンブル・オプション / リストオプション＞

［指定形式］

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

- 省略時解釈

  なし

［詳細説明］

- アセンブラが出力するリスト内容の設定を行います。各指定をした場合に出力される内容は以下の通りです。

表 A—12　show オプション指定一覧

出力種別	内容
conditionals	条件アセンブルで条件が偽となった行もアセンブルリストファイルに出力します。
definitions	.DEFINE で置き換える以前の情報でアセンブルリストファイルに出力します。
expansions	マクロ記述展開行をアセンブルリストファイルに出力します。

マイコンオプション

<アセンブル・オプション / マイコンオプション>

マイコンオプションには，次のものがあります。

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

-isa

<アセンブル・オプション / マイコンオプション>

[指定形式]

```
-isa = { rxv1 | rxv2 }
```

- 省略時解釈

  環境変数 ISA_RX の内容に従います。

[詳細説明]

- 生成する命令コードの命令セットアーキテクチャ (RXv1 または RXv2) を指定します。
- -isa=rxv1 を指定した場合，RXv1 向けのリロケータブルファイルを生成します。
- -isa=rxv2 を指定した場合，RXv2 向けのリロケータブルファイルを生成します。

[備考]

- 選択した命令セットアーキテクチャでサポートされない命令の記述はエラーとなります。
- -nofpu と -fpu のいずれの選択もない場合に isa オプションを指定すると、fpu が自動的に選択されます。
- -isa オプションを省略し、-cpu オプション、環境変数 CPU_RX および 環境変数 ISA_RX のいずれも指定もない場合は、エラーとなります。
- -cpu オプションと同時に指定することはできません。

-cpu

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈

  環境変数 CPU_RX に従います。

［詳細説明］

- マイコン種別を指定します。
- -cpu=rx600 を指定した場合，RX600 向けのリロケータブルファイルを生成します。
- -cpu=rx200 を指定した場合，RX200 向けのリロケータブルファイルを生成します。

［備考］

- 本オプションは、従来機能と互換性を保つためのものです。
- 今後、RX ファミリの製品展開で追加される品種については、命令セットアーキテクチャなどの選択は、-cpu オプションではなく、-isa オプションでの対応となります。新規にアプリケーション開発する際は、-isa オプションを用いてください。
- cpu オプションは、次の記述により -isa オプションと -fpu,-nofpu オプションで置き換えることが可能です。*1
  - - cpu=rx600 → -isa=rxv1 -fpu
  - - cpu=rx200 → -isa=rxv1 -nofpu
- rx200 を指定した場合，RX200 でサポートされていない浮動小数点演算命令の記述および制御レジスタ FPSW の記述をエラーとします。
- -cpu オプションを省略し、-isa オプション、環境変数 CPU_RX および 環境変数 ISA_RX のいずれの指定もない場合はエラーとなります。
- -isa オプションと同時に指定することはできません。

【注意】

- *1) ソースプログラムにプリデファインドマクロ __RX200、__RX600 の記述がある場合を除きます。

-endian

<アセンブル・オプション / マイコンオプション>

［指定形式］

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

- 省略時解釈

  -endian=little です。

［詳細説明］

- endian=big を指定した場合，データのバイト並びが BigEndian になります。
- endian=little を指定した場合，データのバイト並びが LittleEndian になります。

-fint_register

<アセンブル・オプション / マイコンオプション>

[指定形式]

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

- 省略時解釈

  fint_register=0 です。

[詳細説明]

- コンパイラの同名のオプションで指定された，高速割り込み専用で使用する汎用レジスタの情報を，リロケータブルファイルに出力します。

[備考]

- 本オプションは，プロジェクト全体で指定を統一してください。指定が異なる場合の動作は保証しません。
- 高速割り込み専用に指定した汎用レジスタを，アセンブリ言語ファイルにおいて，高速割り込み以外の用途で使用しないでください。使用した場合の動作は保証しません。
- 本オプションの指定の対象となるレジスタが，base オプションで指定されていた場合，エラーとなります。

-base

＜アセンブル・オプション / マイコンオプション＞

[指定形式]

```
-base = {  rom=< レジスタ >
         | ram=< レジスタ >
         | < アドレス値 > = < レジスタ >}
                          < レジスタ >:= {R8 ～ R13}
```

- 省略時解釈

  なし

[詳細説明]

- コンパイラの同名のオプションで指定された，ベースアドレスとして固定で使用する汎用レジスタの情報を，リロケータブルファイルに出力します。

[備考]

- 本オプションは，プロジェクト全体で指定を統一してください。指定が異なる場合の動作は保証しません。
- 本オプションで指定した汎用レジスタをベースレジスタ以外の用途に使用しないでください。使用した場合の動作は保証しません。
- 異なる領域に対して同じ汎用レジスタを指定した場合はエラーとなります。
- fint_register オプションで指定した汎用レジスタを本オプションで指定した場合はエラーとなります。

-patch

<アセンブル・オプション / マイコンオプション>

［指定形式］

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

- 省略時解釈

  なし

［詳細説明］

- CPU の品種ごとに特有の問題を回避します。
- -patch=rx610 を指定した場合，RX610 グループで問題となる MVTIPL 命令を未定義命令として扱います。

  MVTIPL は命令と認識されずにエラーメッセージ E0552113 が出力されます。

-pic

<アセンブル・オプション / マイコンオプション>

[指定形式]

```
-pic
```

- 省略時解釈

  PIC 機能が有効ではない状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。

[詳細説明]

- PIC 機能が有効な状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。

[備考]

- 本オプションに矛盾するコードをアセンブリコードに記述しても，チェックされません。
- PIC 機能が有効なリロケータブルオブジェクトは，PIC 機能が有効でないリロケータブルオブジェクトとはリンクできません。
- PIC 機能については，「PIC/PID 機能の利用」の項目も参照してください。

-pid

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈

  PID 機能が有効ではない状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。

［詳細説明］

- PID 機能が有効な状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。
- ＜PID レジスタ＞

  下記の表の規則に基づき，fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお，fint_register の指定がない場合は R13 を選択します。

表 A—13　fint_register オプションと PID レジスタの関係

fint_register オプション	PID レジスタ
fint_register 指定なし	R13
fint_register=0	
fint_register=1	R12
fint_register=2	R11
fint_register=3	R10
fint_register=4	R9

- PID レジスタは，PID のアクセスに使用する用途以外には使用されません。
- ＜パラメータ＞

  パラメータの意味は，コンパイラの同名オプションと同じです。

［備考］

- 本オプションに矛盾するコードをアセンブリコードに記述しても，チェックされません。
- PID 機能が有効なリロケータブルオブジェクトは，PID 機能が有効でないリロケータブルオブジェクトとはリンクできません。
- pid オプションで選択された PID レジスタが，base オプションでも選択された場合はエラー F0553111 になります。
- pid オプションと nouse_pid_register オプションを同時に選択するとエラー F0553103 になります。
- PID 機能については，「PIC/PID 機能の利用」の項目も参照してください。

-nouse_pid_register

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

```
-nouse_pid_register
```

- 省略時解釈

  なし

［詳細説明］

- PID レジスタを使用しないでコード生成されたことを表すリロケータブルオブジェクトを生成します。
- アセンブリソースファイル中で PID レジスタを使用している場合に，エラーとして E0552058 メッセージを出力します。なお、PID レジスタの名称がアセンブラ言語仕様の「代用レジスタ名」と同一の場合は、本オプションを指定してもエラーにはなりません。
- PID 機能が有効なアプリケーションプログラムから呼び出されるマスタプログラムは，本オプションでアセンブルする必要があります。このとき，アプリケーションに fint_register オプションの選択がある場合は，マスタプログラムにも同じパラメータの fint_register を選択してください。

［備考］

- nouse_pid_register オプションと pid オプションを同時に選択するとエラー F0553103 になります。
- nouse_pid_register オプションで選択されたレジスタが，base オプションでも選択された場合はエラー F0553112 になります。
- PID 機能の詳細については，「PIC/PID 機能の利用」の項目を参照してください。

その他オプション

＜アセンブル・オプション / その他オプション＞

その他オプションには，次のものがあります。

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

-logo

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-logo
```

- 省略時解釈

  コピーライト表示が出力されます。

［詳細説明］

- コピーライト表示が出力されます。

-nologo

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-nologo
```

- 省略時解釈

  コピーライト表示が出力されます。

［詳細説明］

- コピーライトの表示の出力が抑止されます。

-subcommand

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-subcommand=< ファイル名 >
```

- 省略時解釈

  なし

［詳細説明］

- subcommand オプション指定時は，アセンブラ起動時のアセンブラオプションをサブコマンドファイルで指定します。サブコマンドファイル中の書式は，コマンドラインの書式と同一です。

［例］

- ＜サブコマンドファイル opt.sub の内容＞

```
-listfile
-debug
```

- ＜コマンドライン指定＞

  (1) のコマンドライン指定を行うと，アセンブラで (2) のように解釈されます。

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

-

-euc

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-euc
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を sjis コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を euc コードで扱います。

-sjis

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-sjis
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を sjis コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を sjis コードで扱います。

-latin1

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-latin1
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を sjis コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を latin1 コードで扱います。

-big5

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-big5
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を BIG5 コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を big5 コードで扱います。

-gb2312

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-gb2312
```

- 省略時解釈

  文字列，文字定数およびコメント内の文字を GB2312 コードで扱います。

［詳細説明］

- 文字列，文字定数およびコメント内の文字を gb2312 コードで扱います。

(3) 最適化リンケージエディタ（rlink）・オプション

分類	オプション	説明
入力オプション	-Input	リロケータブル・ファイルを指定します。
	-library	ライブラリ・ファイルを指定します。
	-binary	バイナリ・ファイルを指定します。
	-define	シンボル定義を指定します。
	-entry	エントリシンボル / アドレスを指定します。
	-noprelink	プレリンカを起動しません。
出力オプション	-form	出力ファイル形式を選択します。
	-debug	デバッグ情報をロード・モジュール・ファイル内に出力します。
	-sdebug	デバッグ情報を .dbg ファイルに出力します。
	-nodebug	デバッグ情報を出力しません。
	-record	レコードサイズを選択します。
	-rom	ROM から RAM へマップするセクションを指定します。
	-output	出力ファイル名を指定します。
	-map	外部シンボル割り付け情報ファイルを出力します。
	-space	出力範囲のメモリの空き領域をデータで充填します。
	-message	インフォメーションレベル・メッセージの出力します。
	-nomessage	出力抑止メッセージを指定します。
	-msg_unused	参照されない外部定義シンボルをメッセージ出力します。
	-byte_count	データ・レコードのバイト数を指定します。
	-crc	CRC コードの出力形式を指定します。
	-padding	終端にパディングを埋め込みます。
	-vectn	可変ベクタの特定ベクタ番号へのアドレスを指定します。（RX ファミリ、M16C ファミリ向け）
	-vect	可変ベクタの空き領域へのアドレスを指定します。（RX ファミリ、M16C ファミリ向け）
	-jump_entries_for_pic	ジャンプ・テーブル・ファイルを出力します。（RX ファミリの PIC 機能向け）
リストオプション	-list	リンケージ・リスト・ファイルを出力します。
	-show	リンケージ・リスト・ファイルの出力する内容を選択します。

分類	オプション	説明
最適化オプション	-optimize	リンク時最適化内容を選択します。
	-nooptimize	リンク時最適化を行いません。
	-samesize	共通コード統合の対象となる最小サイズを指定します。
	-symbol_forbid	未参照シンボル削除抑止シンボルを指定します。
	-samecode_forbid	共通コード統合抑止シンボルを指定します。
	-section_forbid	最適化抑止セクションを指定します。
	-absolute_forbid	最適化抑止アドレス範囲を指定します。
セクションオプション	-start	セクションの開始アドレスを指定します。
	-fsymbol	外部定義シンボルをファイル出力するセクションを指定します。
	-aligned_section	セクション・アライメントを 16 バイトにします。
ベリファイオプション	-cpu	アドレスの整合性をチェックします。
	-contiguous_section	セクション分割の対象外セクションを指定します。
その他オプション	-s9	S9 レコードを終端に出力します。
	-stack	スタック使用量情報ファイルを出力します。
	-compress	デバッグ情報を圧縮します。
	-nocompress	デバッグ情報を圧縮しません。
	-memory	リンク時に使用するメモリ量を選択します。
	-rename	変更するシンボル名、セクション名を指定します。
	-delete	削除するシンボル名、モジュール名を指定します。
	-replace	置換するライブラリ・モジュール指定します。
	-extract	ライブラリファイルから抽出するモジュールを指定します。
	-strip	アブソリュートファイル、ライブラリファイルのデバッグ情報を削除します。
	-change_message	変更するインフォメーション、ウォーニング、エラーレベルのメッセージレベル、メッセージを指定します。
	-hide	ローカルシンボル名情報を削除します。
	-total_size	リンク後の合計セクションサイズを標準出力に表示します。
サブコマンドファイルオプション	-subcommand	コマンドオプションを取り込むファイルを指定します。
残りのオプション	-logo	コピーライトを出力します。
	-nologo	コピーライトの出力しません。
	-end	END より前に指定したオプション列を実行します。
	-exit	オプション指定の終了を指定します。

入力オプション

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

入力オプションには，次のものがあります。

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

-Input

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-Input = <サブオプション>[{,|△}…]
         <サブオプション> : <ファイル名>[(<モジュール名>[,…])]
```

- 省略時解釈

  なし

［詳細説明］

- 入力ファイルを指定します。複数ある場合にはカンマ（,）または空白文字で区切って指定します。
- ワイルドカード（*,?）も指定できます。ワイルドカードで指定した文字列はアルファベット順に展開します。数字と英文字は数字が先，英大文字と英小文字は英大文字が先になります。
- 入力ファイルとして指定できるのは，コンパイラ，アセンブラ出力オブジェクトファイル，最適化リンケージエディタ出力のリロケータブルファイルおよびアブソリュートファイルです。またライブラリ名 (< モジュール名 >) の形式で，ライブラリ内モジュールを入力ファイルとして指定することもできます。モジュール名は拡張子なしで指定します。
- 入力ファイル名に拡張子の指定がない場合，モジュール名がない時は「obj」，モジュール名がある時は「lib」を仮定します。

［例］

```
input=a.obj lib1(e)                ; a.obj と lib1.lib 内のモジュール e を入力します。
input=c*.obj                       ; c で始まる拡張子 obj のファイルをすべて入力します。
```

［備考］

- form=object および extract 指定時，本オプションは無効です。
- コマンドライン上で入力ファイルを指定する場合は，input 無しで指定します。

-library

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-library= < ファイル名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- ライブラリファイルを指定します。複数ある場合にはカンマ（,）で区切って指定します。
- ワイルドカード（*,?）も指定できます。ワイルドカードで指定した文字列はアルファベット順に展開します。数字と英文字は数字が先，英大文字と英小文字は英大文字が先になります。
- 入力ファイル名に拡張子の指定がない場合は，「lib」を仮定します。
- form=library オプションまたは extract オプション指定時は，ライブラリファイルを編集対象ライブラリとして入力します。
- それ以外の場合は，入力ファイルとして指定されたファイル間でのリンケージ処理後に，未定義シンボルをライブラリファイルから検索します。
- ライブラリファイル内シンボルの検索は，ライブラリオプション指定ユーザライブラリファイル（指定順），ライブラリオプション指定システムライブラリファイル（指定順），デフォルトライブラリ（環境変数 HLNK_LIBRARY1,2,3）の順序で行います。

［例］

```
library=a.lib,b             ; a.lib と b.lib を入力します。
library=c*.lib              ; c で始まる拡張子 lib のファイルをすべて入力します。
```

-binary

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-binary = <サブオプション>[,…]
          <サブオプション> :
          <ファイル名>(<セクション名>[:<アライメント数>][/<セクション属性>][,<シンボル名>])
                      <セクション属性> : CODE | DATA
                      <アライメント数> : 1 | 2 | 4 | 8 | 16 | 32 (デフォルトは1)
```

- 省略時解釈

  なし

［詳細説明］

- 入力バイナリファイルを指定します。複数ある場合にはカンマ（,）で区切って指定します。
- ファイル名に拡張子の指定がない場合は，「bin」を仮定します。
- 入力したバイナリデータは，指定したセクションのデータとして配置します。セクションのアドレスは start オプションで指定します。セクションは省略できません。
- またシンボルを指定することにより，定義シンボルとしてリンクすることもできます。C/C++ プログラムで参照している変数名の場合，プログラム中での参照名先頭に _ を付加します。
- 本オプションで指定したセクションには，セクション属性，アライメント数の指定が可能です。
- セクション属性は，CODE または DATA を指定できます。
- セクション属性の指定が無い場合，デフォルトとして書き込み，読み取り，実行すべての属性が有効になります。
- アライメント数に指定可能な値は 2 の累乗です。それ以外の値を指定することはできません。
- アライメント数の指定がない場合，デフォルト値として 1 が有効になります。

［例］

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

- b.bin を D1bin セクションとして，0x200 番地から配置します。
- c.bin を D2bin セクション（アライメント数 4）として，D1bin の後に配置します。
- c.bin データを定義シンボル _datab としてリンクします。

［備考］

- form={object | library} または strip 指定時，本オプションは無効です。
- また入力オブジェクトファイル指定がない場合，本オプションは指定できません。

-define

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-define = ＜サブオプション＞[,…]
          ＜サブオプション＞ : ＜シンボル名＞ = {＜シンボル名＞ | ＜数値＞}
```

- 省略時解釈

  なし

［詳細説明］

- 未定義シンボルを外部定義シンボルまたは数値で強制定義します。
- 数値は 16 進数で指定します。先頭が A ～ F の場合は先にシンボルを検索し，該当するシンボルがなければ数値として解釈します。先頭に 0 を付加した場合は常に数値と解釈します。シンボル名が C/C++ 変数名の場合，プログラム中での定義名先頭に _ を付加します。C++ 関数名の場合は（main 関数は除く）引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし，引数が void の場合は，" 関数名 ()" で指定します。

［例］

```
define=_sym1=data           ;_sym1 を外部定義シンボル data と同値として定義します。
define=_sym2=4000           ;_sym2 を 0x4000 として定義します。
```

［備考］

- form={object | relocate | library} 指定時，本オプションは無効です。

-entry

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-entry = {< シンボル名 > | < アドレス >}
```

- 省略時解釈

  なし

［詳細説明］

- 実行開始アドレスを外部定義シンボルまたはアドレスで指定します。
- アドレスは 16 進数で指定します。先頭が A ～ F の場合は先に定義シンボルを検索し，該当するシンボルがなければアドレスと判断します。先頭に 0 を付加した場合は常にアドレスと解釈します。
- シンボル名は，C 関数名の場合プログラム中での定義名先頭に _ を付加します。C++ 関数名の場合は（main 関数は除く）引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし，引数が void の場合は，" 関数名 ()" で指定します。
- コンパイル，アセンブル時に entry シンボルを指定している場合，本オプション指定を優先します。

［例］

```
entry=_main           ;C/C++ の main 関数を実行開始アドレスとして設定します。
entry="init()"        ;C++ の init 関数を実行開始アドレスとして設定します。
entry=100             ;0x100 を実行開始アドレスとして設定します。
```

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。
- 未参照シンボル削除最適化（optimize=symbol_delete）指定時には，実行開始アドレスは必ず必要です。指定がない場合は，未参照シンボル削除最適化指定は無効です。本オプションでアドレスを指定している場合は，未参照シンボル削除最適化を無効にします。

-noprelink

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-noprelink
```

- 省略時解釈

  プレリンカを起動します。

［詳細説明］

- プレリンカの起動を抑止します。
- プレリンカは，C++ テンプレートインスタンスの自動生成機能および実行時型検査機能をサポートします。C++ テンプレート機能および実行時型検査機能を使用していない場合は，noprelink オプションを指定してください。リンク時間が短くなります。

［備考］

- extract または strip 指定時，本オプションは無効です。
- C++ テンプレート機能および実行時型検査機能を使用し，form=lib または，form=rel を指定する場合には，noprelink を指定しないでください。

出力オプション

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

出力オプションには，次のものがあります。

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

-form

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

- 省略時解釈

  form=absolute です。

［詳細説明］

- 出力形式を指定します。
- サブオプションの一覧を表 B.13 に示します。

表 A—14　form オプションのサブオプション一覧

No	サブオプション名	内 容
1	absolute	アブソリュートファイルを出力します。
2	relocate	リロケータブルファイルを出力します。
3	object	オブジェクトファイルを出力します。extract オプションでライブラリから 1 個のモジュールをオブジェクトファイルとして取り出すときに使用します。
4	library	ライブラリファイルを出力します。 library=s 指定時，出力ライブラリファイルをシステムライブラリとします。 library=u 指定時，出力ライブラリファイルをユーザライブラリとします。 省略時解釈は，library=u です。
5	hexadecimal	インテル HEX 形式ファイルを出力します。インテル HEX フォーマットは「インテル HEX 形式ファイル」を参照してください。
6	stype	モトローラ S 形式ファイルを出力します。モトローラ S フォーマットは「モトローラ S 形式ファイル」を参照してください。
7	binary	バイナリファイルを出力します。

［備考］

出力形式と入力ファイル，他オプションとの関係を表 B-14 に示します。

表 A—15 出力形式と入力ファイル，他オプションとの関係

No	出力形式	指定オプション	入力可能なファイル形式	指定可能なオプション注 1
1	Absolute	strip あり	アブソリュートファイル	input, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	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,total_size, vector, jump_entries_for_pic, aligned_section
2	Relocate	extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, debug/nodebug, output, hide, rename, delete, noprelink, msg_unused, show=symbol,xreference,total_size
3	Object	extract あり	ライブラリファイル	library, output
4	Hexadecimal Stype Binary		オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	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 注 2, byte_count 注 3, memory, msg_unused, show=symbol,reference,xreference,total_size, vector, jump_entries_for_pic, aligned_section
			アブソリュートファイル	input, output, record, s9 注 2, byte_count 注 3, show=symbol, reference, xreference
5	Library	strip あり	ライブラリファイル	library, output, memory 注 4, show=symbol, section
		extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル	input, library, output, hide, rename, delete, replace, noprelink, memory 注 4, show=symbol, section

注 1. message/nomessage, change_message, logo/nologo, form, list, subcommand は常に指定できます。

2. s9 は出力形式が form=stype のときだけ指定できます。

3. byte_count は出力形式が form= hexadecimal のときだけ指定できます。

4. hide 指定する場合は使用できません。

-debug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-debug
```

- 省略時解釈

  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

- 出力ファイル中にデバッグ情報を出力します。
- output オプションで複数ファイル出力を指定時に debug オプションを指定したときは，sdebug オプションと解釈して，< 先頭出力ファイル名 >.dbg に出力します。

［備考］

- form={object | library | hexadecimal | stype | binary}，strip，または，extract 指定時，本オプションは無効です。

-sdebug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-sdebug
```

- 省略時解釈

  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

- < 出力ファイル名 >.dbg ファイルにデバッグ情報を出力します。
- form=relocate 指定時に sdebug オプションを指定したときは，debug オプションと解釈します。

［備考］

- form={object | library | hexadecimal | stype | binary}，strip，または，extract 指定時，本オプションは無効です。

-nodebug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-nodebug
```

- 省略時解釈

  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

- デバッグ情報を出力しません。

［備考］

- form={object | library | hexadecimal | stype | binary}，strip，または，extract 指定時，本オプションは無効です。

-record

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

- 省略時解釈

  それぞれのアドレスに合わせて混在したデータレコードを出力します。

［詳細説明］

- アドレス範囲に関係なく，一定のデータレコードで出力します。
- 指定したデータレコードより大きいアドレスが存在した場合，アドレスに合わせてデータレコードを選択します。

［備考］

- form=hexadecimal または stype 指定がないとき，本オプションは無効です。

-rom

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-rom = <サブオプション>[,…]
       <サブオプション> : <ROM セクション名>=<RAM セクション名>
```

- 省略時解釈

  なし

［詳細説明］

- 初期化データ領域の ROM 用，RAM 用領域を確保し，ROM セクション内定義シンボルを RAM セクション内アドレスになるようリロケーションします。
- ROM セクションには初期値のあるリロケータブルセクションを指定します。
- RAM セクションには存在しないセクションまたはサイズ 0 のリロケータブルセクションを指定します。

［例］

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

- D セクションと同サイズの R セクションを確保し，D セクション内定義シンボルを R セクション上のアドレスでリロケーションします。

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。

-output

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-output ＝ ＜サブオプション＞[,…]
          ＜サブオプション＞ ：  ＜ファイル名＞[=＜出力範囲＞]
          ＜出力範囲＞: {＜先頭アドレス＞-＜終了アドレス＞ | ＜セクション名＞[:…]}
```

- 省略時解釈

  ＜先頭入力ファイル名＞.＜デフォルト拡張子＞です。

- デフォルト拡張子は，次のようになります。

  form=absolute：「abs」，form=relocate　：「rel」，form=object：「obj」，form=library ：「lib」，form=hexadecimal：「hex」，form=stype ：「mot」，form=binary ：「bin」

［詳細説明］

- 出力ファイル名を指定します。form={absolute | hexadecimal | stype | binary} のときは，複数ファイルを指定できます。アドレスは 16 進数で指定します。先頭が A ～ F の場合は先にセクションを検索し，該当するセクションがなければアドレスと判断します。先頭に 0 を付加した場合は常にアドレスと解釈します。

［例］

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

- 0 ～ 0xffff 間を file1.abs に，0x10000 ～ 0x1ffff 間を file2.abs に出力します。

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

- sec1,sec2 セクションを file1.abs に，sec3 セクションを file2.abs に出力します。

［備考］

- マイコン種別が RX ファミリでビッグエンディアンのときに，セクション単位で出力する場合は，セクションサイズを 4 の倍数にしてください。

-map

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-map [= < ファイル名 >]
```

- 省略時解釈

  なし

［詳細説明］

- コンパイラが外部変数アクセス最適化で使用する外部変数割り付け情報ファイルを出力します。
- < ファイル名 > を指定しなかった場合は，output オプションで指定したファイル名，もしくは先頭入力ファイル名で，拡張子が bls のファイルを出力します。
- 外部変数割り付け情報ファイル作成時の変数宣言順と，再コンパイル後のオブジェクトを読み込んだ時の変数宣言順が変わっている場合はエラーを出力します。

［備考］

- form={absolute | hexadecimal | stype | binary} を指定した場合のみ，本オプションは有効です。

-space

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-space [ = {< 数値 > | Random}]
```

- 省略時解釈

  なし

［詳細説明］

- 出力範囲のメモリの空き領域を，ユーザが指定するデータで充填します。
- 充填するデータとしては，乱数，もしくは 16 進数の数値を指定することができます。
- 空きエリアを埋める方法は，output オプション指定時の出力範囲指定方法によって下記のように異なります。
  - 出力範囲 : セクション指定
    - 指定されたセクション間に空きが存在した場合に指定データを出力
  - 出力範囲 : アドレス範囲指定
    - 指定された範囲内に空きが存在した場合に指定データを出力
- 出力データサイズは，1,2,4 バイト単位で有効となります。出力データサイズは space オプションに指定する 16 進数の数値で決まります。3 バイトデータを指定した場合，上位桁を 0 拡張し 4 バイトのデータとして扱われます。また，奇数桁データを指定した場合も，上位桁に 0 拡張して偶数桁入力として扱われます。
- 空きエリアのサイズが出力データサイズの倍数でない場合，出力できるだけ出力し，メッセージによる警告を行います。

［備考］

- 本オプションにてサブオプションの指定がされなかった場合は，空きエリアへの出力は行いません。
- 本オプションは form={ binary |stype | hexadecimal} オプションを指定した場合にのみ有効となります。
- output オプションによる出力範囲指定がされなかった場合は，本オプション指定は無効となります。

-message

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-message
```

- 省略時解釈

  nomessage (インフォメーションレベルメッセージの出力を抑止) です。

［詳細説明］

- インフォメーションレベルメッセージを出力します。

-nomessage

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-nomessage [= <サブオプション>[,…]]
             <サブオプション> : <エラー番号>[-<エラー番号>]
```

- 省略時解釈

  nomessage (インフォメーションレベルメッセージの出力を抑止) です。

［詳細説明］

- インフォメーションレベルメッセージの出力を抑止します。またエラー番号を指定すると，指定したエラー番号のメッセージ出力を抑止できます。ハイフン（-）を使用して抑止するエラー番号の範囲を指定することもできます。エラー番号としてウォーニング，エラーレベルメッセージ番号を指定した場合，change_message でインフォメーションレベルに変更したと仮定し，メッセージ出力を抑止します。
- エラー番号には，コンポーネント番号（05），発生フェーズ（6）に続けて出力される 4 桁の数値（M0560004 の場合は 0004）を指定します。4 桁の数値の先頭が 0 で始まる場合は，0 を省略することができます（M0560004 の場合は 4）。

［例］

- M0560004、M0560200 ～ M0560203 および M0561300 のメッセージ出力を抑止します。

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

-msg_unused

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-msg_unused
```

- 省略時解釈

  なし

［詳細説明］

- 本オプションを指定した場合，リンク処理の中で一度も参照されることのなかった外部定義シンボルを，メッセージ出力によってユーザに知らせます。

［例］

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

［備考］

- 入力ファイルが absolute 形式の場合，本オプション指定は無効です。
- メッセージ出力させるためには，同時に message オプションの指定が必要です。
- コンパイル時にインライン展開された関数に対してメッセージ出力する場合があります。その場合，関数定義に static 宣言することで，メッセージ出力を抑えることができます。
- 以下のいずれかに該当する場合，参照関係の解析が正しく行うことができず，メッセージ出力により通知される情報が不正確となります。
  - 同一ファイル内の定数シンボルへの参照
  - コンパイル時に最適化が有効で，直下の関数を呼び出す場合

-byte_count

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-byte_count=< 数値 >
```

- 省略時解釈

  バイト数最大値は FF として Intel-Hex ファイルを生成します。

［詳細説明］

- Intel-Hex 形式ファイルを生成する際に，データレコードのバイト数最大値を指定するためのオプションです。
- バイト数としては，1byte の 16 進数（01 ～ FF）を指定することができます。

［例］

```
byte_count=10
```

［備考］

- 生成するファイル形式が Intel-Hex 形式（form=hex）ではない場合，本オプションは無効です。

-crc

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-CRc = <サブオプション>
       <サブオプション>: <出力位置>=<計算範囲>[/<多項式>][:<エンディアン>]
                         <出力位置>: <アドレス>
                         <計算範囲>: <先頭アドレス>-<終了アドレス>[,...]
                         <多項式>  : { CCITT | 16 }
                         <エンディアン>: {BIG|LITTLE}
```

- 省略時解釈

  なし

［詳細説明］

- 計算範囲で指定された内容を下位アドレスから上位アドレスの順で CRC（Cyclic Redundancy Check）演算を行い，計算結果を出力位置のアドレスに出力します。
- エンディアンは，RX ファミリの場合に指定可能なオプションです。エンディアンを指定した場合は，エンディアンにしたがって，計算結果を出力位置のアドレスに出力します。指定しない場合は，アブソリュートファイルのエンディアンで計算結果を出力位置のアドレスに出力します。
- 多項式は CRC-CCITT または CRC-16 を選択できます。（デフォルトは CRC-CCITT）
- 多項式

```
CRC-CCITT
    X^16+X^12+X^5+1
        ビット表現（10001000000100001）
CRC-16
    X^16+X^15+X^2+1
        ビット表現（11000000000000101）
```

［例］

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

  - crc オプション：-crc=2FFE=1000-2FFD
    - 0x1000 ～ 0x2FFD の領域に対して CRC 演算を行い，その結果を 0x2FFE 番地に出力します。
    - 計算範囲にある空き領域は space オプションが指定されていない場合，space=0xFF が指定されていると仮定して，CRC 演算を行います。

- output オプション：-output=out.mot=1000-2FFF
  - space オプションが指定されていないため，空きの領域は「out.mot」ファイルに出力されません。CRC 演算は，空き領域では 0xFF で計算を行いますが，0xFF を埋めることはありません。

注 1. CRC 出力位置は，計算範囲に含むことは出来ません。
2. CRC 出力位置は output オプションの出力範囲に含まれている必要があります。

- 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 オプション：-crc=2FFE=1000-2FFD,2000-27FF
  - 0x1000 ～ 0x17FF と 0x2000 ～ 0x27FF の２つの領域に対して CRC 演算を行い，その結果を 0x2FFE 番地に出力します。
  - CRC 演算は計算対象として，連続していない複数の計算範囲を指定できます。
- space オプション：-space=7F
  - 指定された計算範囲の空き領域は space オプションの値（0x7F）で計算されます。
- output オプション：-output=out.mot=1000-2FFF
  - space オプションが指定されているため，空き領域は「out.mot」ファイルに出力されます。空き領域は 0x7F で充填されます。

注 1. CRC 演算の計算順は計算範囲の指定順ではありません。下位アドレスから上位アドレスの順に計算されます。
2. crc オプションと space オプションを同時に指定する場合，space オプションに random または２バイト以上の値を指定することは出来ません。1 バイトのデータを指定してください。

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

- crc オプション：-crc=1FFE=1000-1FFD,2000-2FFF
  - 0x1000 ～ 0x1FFD と 0x2000 ～ 0x2FFF の領域に対して CRC 演算を行い，その結果を 0x1FFE 番地に出力します。
  - 計算範囲にある空き領域は space オプションが指定されていない場合，space=0xFF が指定されていると仮定して，CRC 演算を行います。
- output オプション：-output=flmem.mot=1000-1FFF
  - space オプションが指定されていないため，空きの領域は「flmem.mot」ファイルに出力されません。
  - CRC 演算は，空き領域では 0xFF で計算を行いますが，0xFF を埋めることはありません。

［備考］

- 複数のアブソリュートファイル入力時は，本オプションは無効です。
- 出力形式が form={hexadecimal | stype} の場合に有効です。
- space オプションが指定されていない場合で，計算範囲に出力されない空き領域があるとき，空き領域には 0xFF が設定されているものとして CRC の計算が行われます。
- CRC 演算の計算範囲にオーバーレイ指定されている領域が含まれる場合はエラーになります。
- サンプルコード
  - crc オプションで計算された CRC 演算結果を比較するためのサンプルコードです。
  - サンプルコードのプログラムは，rlink の CRC 演算結果と一致します。
- 多項式 CRC-CCITT の場合（初期値 0xFFFF, MSB ファースト , 反転出力）

```
typedef  unsigned   char    uint8_t;
typedef  unsigned   short   uint16_t;
typedef  unsigned   long    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;
}
```

- 多項式 CRC-16 の場合（初期値 0x0000, LSB ファースト , 非反転出力）

```
#define  POLYNOMIAL  0xa001      // 生成多項式 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

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-padding
```

- 省略時解釈

  なし

［詳細説明］

- セクションサイズが，セクションのアライメントの倍数となるように，セクション終端にデータを埋め込みます。

［例］

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

P セクションのアライメント :4 バイト

P セクションのサイズ :0x06 バイト

C セクションのアライメント :1 バイト

C セクションのサイズ :0x03 バイト

の場合，

P セクションに 2 バイトのパディングデータを埋め込んで，サイズを 0x08 バイトにしてリンクする。

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

P セクションのアライメント :4 バイト

P セクションのサイズ :0x06 バイト

C セクションのアライメント :1 バイト

C セクションのサイズ :0x03 バイト

の場合，

P セクションに 2 バイトのパディングデータを埋め込んで，サイズを 0x08 バイトにしてリンクすると，C セクションと重複してしまうため，E0562231 エラーを出力する。

［備考］

- 生成するパディングデータの値は 0x00 です。
- 絶対アドレスセクションには，パディングを行いませんので，絶対アドレスセクションはユーザにてサイズを調整してください。

-vectn

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-vectn = ＜サブオプション＞[,...]
        ＜サブオプション＞ ： ＜ベクタ番号＞ = {＜シンボル＞ | ＜アドレス＞}
```

- 省略時解釈

  なし

［詳細説明］

- 可変ベクタテーブルセクションの特定ベクタ番号に対して，オプションで指定されたアドレスを設定します。
- 本オプションを使用した場合，ソース上に割り込み関数記述がなくても，可変ベクタテーブルセクションを作成し，テーブルへアドレスを設定します。
- ＜ベクタ番号＞は，10 進数で 0 ～ 255 の範囲で指定してください。
- ＜シンボル＞は，対象関数の外部名で指定してください。
- ＜アドレス＞は，指定アドレスを 16 進数で指定してください。

［例］

```
-vectn=30=_f1,31=0000F100          ; ベクタ番号 30 番に _f1 のアドレスを，
                                   ; ベクタ番号 31 番に 0x0f100 を設定します
```

［備考］

ユーザが可変ベクタテーブルセクションをソースプログラムで作成している場合，可変ベクタテーブルの自動生成は行わないため，本オプションは無効になります。

-vect

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-vect={< シンボル >|< アドレス >}
```

- 省略時解釈

  なし

［詳細説明］

- 可変ベクタテーブルセクションで，アドレス未設定のベクタ番号に対してオプション指定のアドレスを設定します。
- 本オプションを使用した場合，ソース上の割り込み関数記述がなくても，可変ベクタテーブルセクションをリンカが作成し，テーブルへアドレスを設定します。
- < シンボル > は，対象関数の外部名を記述してください。
- < アドレス > は，設定するアドレスを 16 進数表記で記述してください。

［備考］

ユーザが可変ベクタテーブルセクションをソースプログラムで作成している場合，可変ベクタテーブルの自動生成は行わないため，本オプションは無効になります。

{< シンボル >|< アドレス >} の記述で，先頭を 0 と記述したものはすべてアドレスとして判断します。

-jump_entries_for_pic

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-jump_entries_for_pic = < セクション名 >[,...]
```

- 省略時解釈

  なし

［詳細説明］

- 指定セクション内の外部定義シンボルへ分岐するジャンプテーブルのアセンブラソースを出力します。
- ファイル名は，< 出力ファイル >.jmp です。

［例］

- セクション sct2,sct3 の外部定義シンボルへ分岐するジャンプテーブルを test.jmp に出力します。

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

- [test.jmp の出力例]

```
;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
```

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。
- 生成するジャンプテーブルは，P セクションへ出力します。
- セクション名に指定できるセクション種別は，プログラムセクションのみです。

リストオプション

＜最適化リンケージエディタ（rlink）・オプション / リストオプション＞

リストオプションには，次のものがあります。

- -list
- -show

-list

＜最適化リンケージエディタ（rlink）・オプション / リストオプション＞

［指定形式］

```
-list [= < ファイル名 >]
```

- 省略時解釈

  なし

［詳細説明］

- リストファイル出力およびリストファイル名を指定します。
- リストファイル名を指定しない場合には，出力（または先頭出力）ファイルと同じファイル名で，拡張子が form=library または extract 指定時「lbp」，それ以外のとき「map」のリストファイルが作成されます。

-show

＜最適化リンケージエディタ（rlink）・オプション / リストオプション＞

［指定形式］

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

- 省略時解釈

  なし

［詳細説明］

- リストの出力内容を指定します。
- サブオプションの一覧を表 B-15 に示します。
- 各リストの具体例については「リンケージリスト」,「ライブラリリスト」を参照してください。

表 A—16　show オプションのサブオプション一覧

No	出力形式	サブオプション名	意味
1	form=library	symbol	モジュール内シンボル名一覧を出力します。
		reference	指定できません。
		section	モジュール内セクション一覧を出力します。
		xreference	指定できません。
		total_size	指定できません。
		vector	指定できません。
		all	モジュール内シンボル名，セクション一覧を出力します。
2	form=library 以外	symbol	シンボルアドレス，サイズ，種別，最適化内容を出力します。
		reference	シンボルの参照回数を出力します。（form=relocate の時は指定できません）
		section	指定できません。
		xreference	クロスリファレンス情報を出力します。
		total_size	ROM 配置対象，RAM 配置対象ごとに，セクションの合計サイズを表示します。

No	出力形式	サブオプション名	意味
		vector	ベクタ情報を出力します。（form=relocatable のときは指定できません）
		all	show=symbol,xreferrence,total_size 指定時と同内容を出力します。 （form=rel） show=symbol,reference,xreference ,total_size 指定時と同内容を出力します。 （form=abs） show=symbol,reference,xreference,total_size 指定時と同内容を出力します。 （form=hex/stype/bin） form=obj のときは指定できません。

［備考］

- オプション form とオプション show および show=all で有効 / 無効になる組み合わせは以下のようになります。

		Symbol	Reference	Section	Xreference	Vector	Total_size
form=abs	show のみ	有効	有効	無効	無効	無効	無効
	show=all	有効	有効	無効	有効	有効	有効
form=lib	show のみ	有効	無効	有効	無効	無効	無効
	show=all	有効	無効	有効	無効	無効	無効
form=rel	show のみ	有効	無効	無効	無効	無効	無効
	show=all	有効	無効	無効	有効注	無効	有効
form=obj	show のみ	有効	有効	無効	無効	無効	無効
	show=all	無効	無効	無効	無効	無効	無効
form=hex/ bin/sty	show のみ	有効	有効	無効	無効	無効	無効
	show=all	有効	有効	無効	有効	有効注	有効注

注 入力ファイルが absolute 形式の場合は無効です。
クロスリファレンス情報の出力に関しては，下記制限があります。

- 入力ファイルが absolute 形式の場合，参照側アドレスの情報は出力されません。
- 同一ファイル内の，定数シンボルへの参照に関する情報は出力されません。
- コンパイル時に最適化が有効で，直下の関数を呼び出す場合についての情報は出力されません。
- 外部変数アクセス最適化が有効な場合，ベースとなるシンボルを除いて，変数の参照情報は出力されません。
- show=total_size で表示する情報は，別オプション total_size での表示内容と同じです。
- show=reference が有効な場合に，#pragma address で指定された変数の参照回数が 0 として出力されます。
- extract が指定されている場合、サブオプションを指定することはできません。

最適化オプション

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

最適化オプションには，次のものがあります。

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

-optimize

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-optimize[= <サブオプション>[,…]]
<サブオプション> : {SYmbol_delete | SAMe_code | SHort_format | Branch | SPeed | SAFe}
```

- 省略時解釈

  optimize です。

［詳細説明］

- コンパイル，アセンブル時に goptimize オプションを指定したファイルに対して最適化を行います。
- サブオプションの一覧を表 B-16 に示します。

表 A—17　optimize オプションのサブオプション一覧

サブオプション	意　味	最適化対象プログラム[注 1]	
		RXC	RXA
パラメータなし	すべての最適化を実行します。	○	×
symbol_delete	1 度も参照のない変数 / 関数を削除します。必ずコンパイル 時に #pragma entry を指定するか，rlink で entry オプションを指定してください。	○	×
same_code	複数の同一命令列をサブルーチン化します。	○	×
short_format	ディスプレースメント / イミディエートのコードサイズを短縮可能な場合，コードサイズがより小さくなる命令に置き換えます。	○	×

サブオプション	意 味	最適化対象プログラム注 1	
		RXC	RXA
branch	プログラムの配置情報に基づいて，分岐命令サイズを最適化します。 symbol_delete 以外の他の最適化項目を実行すると，指定の有無に関わらず必ず実行します。	○	×
speed	オブジェクトスピード低下を招く可能性のある最適化以外を実行します。 optimize=symbol_delete,short_format,branch と同じです。	○	×
safe	変数や関数の属性によって制限される可能性のある最適化以外を実行します。 optimize=short_format,branch と同じです。	○	×

注 1. RXC: RX ファミリ用 C/C++ プログラム，RXA: RX ファミリ用アセンブリプログラム

［備考］

- form=｛object|relocate|library｝または strip 指定時，本オプションは無効です。
- プログラム内に #pragma entry で実行開始関数を指定，または実行開始アドレス（entry）を指定していない場合，optimize=symbol_delete は無効になります。

-nooptimize

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-nooptimize
```

- 省略時解釈

  optimize です。

［詳細説明］

- リンク時最適化を行いません。

-samesize

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-samesize = < サイズ >
```

- 省略時解釈

  samesize=1E です。

［詳細説明］

- 共通コード統合最適化（optimize=same_code）で，最適化対象となる最小コードサイズを指定します。8 ～ 7FFF までの 16 進数で指定してください。

［備考］

optimize=same_code の指定がないとき，本オプションは無効です。

-symbol_forbid

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-symbol_forbid = < シンボル名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- 未参照シンボル削除最適化を抑止します。
- C/C++ 変数名，C 関数名はプログラム中での定義名先頭に _ を付加します。C++ 関数の場合は，引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし，引数が void の場合は，" 関数名 ()" で指定します。

［備考］

最適化を使用しないリンク処理では，本オプションは無効です。

-samecode_forbid

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-samecode_forbid = < 関数名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- 共通コード統合最適化を抑止します。
- C/C++ 変数名，C 関数名はプログラム中での定義名先頭に _ を付加します。C++ 関数の場合は，引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし，引数が void の場合は，" 関数名 ()" で指定します。

［備考］

最適化を使用しないリンク処理では，本オプションは無効です。

-section_forbid

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-section_forbid = <sub>[,…]
                  <sub>: [< ファイル名 >|< モジュール名 >](< セクション名 >[,…])
```

- 省略時解釈

  なし

［詳細説明］

- -section_forbid のサブオプションで指定したセクションの最適化を抑止します。入力ファイル名，もしくはライブラリモジュール名を同時に指定することで，最適化抑止対象をセクション全体だけでなく，指定したファイルに限定することも可能です。

［備考］

- 最適化を使用しないリンク処理では，本オプションは無効です。
- パスを記述した入力ファイルを最適化抑止する場合，section_forbid オプションではファイル名にパスを記述してください。

-absolute_forbid

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞

［指定形式］

```
-absolute_forbid = <アドレス>[+ <サイズ>][,…]
```

- 省略時解釈

  なし

［詳細説明］

- -absolute_forbid のサブオプションで指定したアドレス＋サイズの範囲の最適化を抑止します。

［備考］

- 最適化を使用しないリンク処理では，本オプションは無効です。

セクションオプション

＜最適化リンケージエディタ（rlink）・オプション / セクションオプション＞

セクションオプションには，次のものがあります。

- -start
- -fsymbol
- -aligned_section

-start

＜最適化リンケージエディタ（rlink）・オプション / セクションオプション＞

［指定形式］

```
-start= <sub>[,...]
        <sub> : [(]< セクション名 > [{: | ,} < セクション名 >[,...] ][)][,...] [/< アドレス >]
```

- 省略時解釈

  セクションを 0 番地から割り付けます。

［詳細説明］

- セクションの開始アドレスを指定します。アドレスは 16 進数で指定してください。
- セクション名はワイルドカード（*）も指定できます。ワイルドカードで指定したセクションはオブジェクトファイルの入力順に展開します。
- セクションをコロン（:）で区切ることにより，複数のセクションを同一アドレスに割り付ける（セクションオーバーレイ配置）ことが可能です。
- 同一アドレスに割り付け指定したセクション間は，指定順に割り付けます。
- また，丸括弧 "()" で囲むことにより，オーバーレイ配置する対象セクションを変更できます。
- 同一セクション内オブジェクトは，入力ファイルの指定順，入力ライブラリの指定順に割り付けます。
- アドレスの指定がない場合は，0 番地から割り付けます。
- start オプションで指定していないセクションは，最終割り付けアドレスに続いて割り付けます。

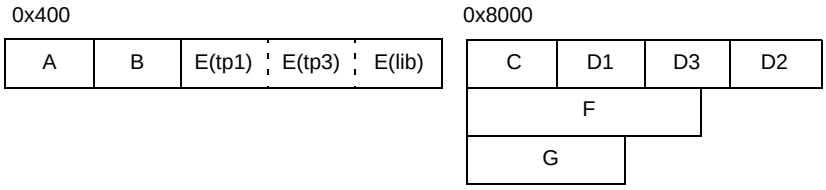
［例］

下記順番でオブジェクトを入力する場合のセクション配置を例に示します。

（括弧内は各オブジェクトが持つセクション）

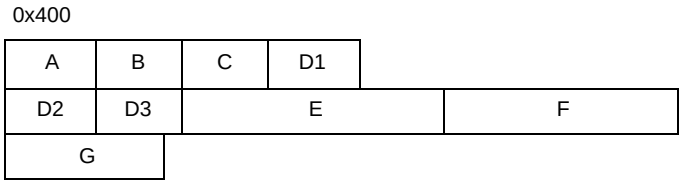
tp1.obj(A,D1,E) -> tp2.obj(B,D3,F) -> tp3.obj(C,D2,E,G) -> lib.lib(E)

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



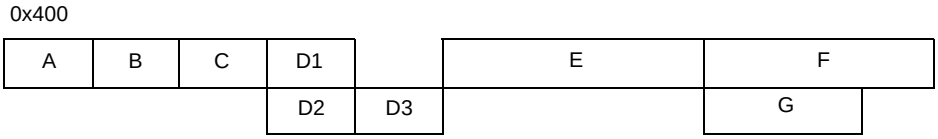
- ":" で区切った C,F,G セクションは，同一アドレスに割り付きます。
- ワイルドカードで記述したセクション（ここでは D で始まる名前のセクション）は，入力した順番で割り付きます。
- 同名セクション内（ここでは E セクション）は，入力したオブジェクトから順番に割り付きます。
- ライブラリ入力による同名セクション（ここでは E セクション）は，入力オブジェクトの次に割り付きます。

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



- ":" で区切った直後のセクション（この例の場合は A,D2,G）を先頭として，それぞれ先頭が同一アドレスに割り付きます。

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



- "()" で同一アドレス配置を括った場合，"()" の直前のセクション（この例の場合は C,E）の直後を先頭として，"()" 内の同一アドレス配置が行われます。
- "()" の直後のセクション（この場合 E）は，"()" 内の最後尾のセクションの直後に続けて配置されます。

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。
- 括弧 "()" は，ネストして記述することはできません。
- 括弧 "()" 内では，少なくともひとつはコロン ":" の記述が必要です。コロン ":" を記述しない場合には，括弧 "()" は記述できません。
- 括弧 "()" を記述した場合，"()" 外にコロン ":" を記述することはできません。
- 括弧 "()" を使用して本オプションを記述した場合，リンカの最適化機能は無効になります。

-fsymbol

＜最適化リンケージエディタ（rlink）・オプション / セクションオプション＞

［指定形式］

```
-fsymbol = < セクション名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- 指定したセクション内外部定義シンボルをアセンブラ制御命令形式でファイルに出力します。

  ファイル名は，< 出力ファイル >.fsy です。

［例］

- セクション sct2,sct3 の外部定義シンボルを test.fsy に出力します。

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

- [test.fsy の出力例]

```
;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
```

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。

-aligned_section

＜最適化リンケージエディタ（rlink）・オプション / セクションオプション＞

［指定形式］

```
-aligned_section = < セクション名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- 指定セクションのアライメント数を 16 byte に変更します。

［備考］

- form={object | relocate | library} および extract，strip 指定時，本オプションは無効です。

ベリファイオプション

＜最適化リンケージエディタ（rlink）・オプション / ベリファイオプション＞

ベリファイオプションには，次のものがあります。

- -cpu
- -contiguous_section

-cpu

＜最適化リンケージエディタ（rlink）・オプション / ベリファイオプション＞

［指定形式］

```
-cpu = { ＜メモリ種別＞ = ＜アドレス範囲＞[,…] | STRIDE}
         ＜メモリ種別＞ = { ROm | RAm | FIX }
         ＜アドレス範囲＞ : ＜先頭アドレス＞ - ＜終了アドレス＞
```

- 省略時解釈

  なし

［詳細説明］

- cpu=stride 未指定時は，セクションの割り付けアドレスに対して，アドレス範囲に入らない場合は，エラーを出力します。
- cpu=stride 指定時は，セクションの割り付けアドレスに対して，アドレス範囲に入らない場合は，次の同メモリ種別に配置，または，分割して配置します。

［例］

- サブオプション stride を指定しない場合

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

  - D1 が 100-1FF，D2 が 200-2FF の範囲に収まるとき，正常終了します。収まらないときエラーを出力します。

- サブオプション stride を指定した場合

```
start=D1,D2/100
cpu=ROM=100-1FF,RAM=200-2FF,ROM=300-3FF
cpu=stride
```

  - D1,D2 が ROM 属性の領域に（セクションを分割して / 分割しないで）収まるとき，正常終了します。セクションを分割しても収まらないときリンクエラーになります。

- セクション割り付けが可能なアドレス範囲を 16 進数で指定してください。ROM/RAM の属性は，モジュール間最適化で使用します。
- メモリ種別“FIX”には，アドレス固定の領域（I/O エリア等）を指定します。
- メモリ種別“FIX”と，それ以外のメモリ種別のアドレス範囲が重複した場合は，メモリ種別“FIX”を有効とします。
- サブオプション stride は，メモリ種別が，ROM または RAM で，アドレス範囲にセクションが収まらなかった場合に，セクションを分割して同じメモリ種別の領域に割り付けます。
- サブオプション stride で，セクションを分割する単位は，モジュール単位になります。

```
cpu=ROM=0-FFFF,RAM=10000-1FFFF
```

- セクションアドレスが，0-FFFF または 10000-1FFFF の間に入っているかチェックします。
- モジュール間最適化では，異なる属性間でのオブジェクトの移動は行いません。

```
cpu=ROM=100-1FF,ROM=400-4FF,RAM=500-5FF
cpu=stride
```

- セクションアドレスが，100-1FF の間に収まらなかった場合に，セクションをモジュール単位で分割して 400-4FF に割り付けます。

［備考］

- form={object | relocate | library} または strip 指定時，本オプションは無効です。
- cpu=stride および memory=low 指定時，無効になります。
- cpu=stride を指定し，B セクションが分割された場合，0 初期化するための情報として 8 バイト×分割数分だけ C$BSEC セクションのサイズが増加します。

-contiguous_section

＜最適化リンケージエディタ（rlink）・オプション / ベリファイオプション＞

［指定形式］

```
-contiguous_section=< セクション名 >[,...]
```

- 省略時解釈

  なし

［詳細説明］

- cpu=stride が有効なときに，セクションを分割せずに同じメモリ種別の割り付け可能なアドレス領域に割り付けるセクションを指定します。

［例］

```
start=P,PA,PB/100
cpu=ROM=100-1FF,ROM=300-3FF,ROM=500-5FF
cpu=stride
contiguous_section=PA
```

- セクション P を 100 番地に割り付けます。
- contiguous_section 指定したセクション PA が，1FF 番地までに割り付けることができない場合，セクション PA を分割せずに，300 番地から割り付けます。
- contiguous_section 指定してないセクション PB が，3FF 番地までに割り付けることができない場合，セクション PB を分割して，500 番地から割り付けます。

［備考］

- cpu オプションのサブオプションの stride が無効なとき，本オプションは無効です。

その他オプション

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

その他オプションには，次のものがあります。

-s9

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-s9
```

- 省略時解釈

  なし

［詳細説明］

- エントリアドレスが 0x10000 を超える場合でも，S9 レコードを終端に出力します。

［備考］

- form=stype 指定がないとき，本オプションは無効です。

-stack

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-stack
```

- 省略時解釈

  なし

［詳細説明］

- スタック使用量情報ファイルを出力します。
- ファイル名は，< 出力ファイル名 >.sni になります。

［備考］

- form={object | relocate | library} および strip 指定時，本オプションは無効です。

-compress

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-compress
```

- 省略時解釈

  デバッグ情報を圧縮しません。

［詳細説明］

- デバッグ情報を圧縮します。
- デバッグ情報を圧縮すると，デバッガのロード速度が速くなります。

［備考］

- form={object | relocate | library | hexadecimal | stype | binary} または strip オプションを指定した場合，compress オプションは無効です。

-nocompress

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-nocompress
```

- 省略時解釈

  デバッグ情報を圧縮しません。

［詳細説明］

- デバッグ情報を圧縮しません。
- nocompress オプションを指定すると，compress オプションを指定したときに比べてリンク時間が短くなります。

-memory

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-memory = [ High | Low ]
```

- 省略時解釈

  memory=high です。

［詳細説明］

- リンク時に使用するメモリ量を指定します。
- memory=high オプションを指定した場合，従来通りの処理を行います。
- memory=low オプションを指定した場合，リンク時に必要な情報のロードを細かく行うことにより，使用するメモリ量の削減を行います。ファイルアクセスの頻度が増えるため，メモリ使用量が実装メモリを超えない状況では memory=high オプション指定より処理が遅くなります。
- 大規模なプロジェクトをリンクした際，最適化リンケージエディタのメモリ使用量が稼動マシンの実装メモリ量を越えてしまい，動作が遅くなっているような場合には memory=low オプション指定をお試しください。

［備考］

- 下記オプションを指定した場合、-memory=low オプション指定は無効となります。
- form=absolute,hexadecimal,stype,binary 指定時

  compress,delete,rename,map,stack,cpu=stride

  list と show[={reference | xreference}] を同時指定
- form=library 指定時

  delete,rename,extract,hide,replace
- form=object,relocate 指定時

  extract
- optimize を指定時

  また，入力ファイルや出力ファイル形式によっても無効となる組み合わせがあります。

-rename

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

[指定形式]

```
-rename = <サブオプション>[,…]
          <サブオプション> : { [<ファイル>](<名前>=<名前>[,…])
                             | [<モジュール>](<名前>=<名前>[,…])}
```

- 省略時解釈

  なし

[詳細説明]

- 外部シンボル名，セクション名を変更します。
- 特定のファイルまたは特定のライブラリ内モジュールに含まれるシンボル名，セクション名を変更することもできます。
- C/C++ 変数名の場合，プログラム中での定義名先頭に _ を付加します。
- 関数名を変更した場合の動作は保証できません。
- 指定した名前がセクション，シンボルの両方に存在した場合，シンボル名を優先します。
- 同一ファイル名，モジュール名が複数存在する場合は，先に入力した方を優先します。

[例]

```
rename=(_sym1=data)             ;_sym1 を data に変更します。
rename=lib1(P=P1)               ; ライブラリモジュール lib1 内の P セクションを
                                ;P1 セクションに変更します。
```

[備考]

- extract または strip 指定時，本オプションは無効です。
- form=absolute 指定時，入力されたライブラリのセクション名を変更することができません。
- コンパイル・オプション -merge_files と本オプションを組み合わせて使用した場合，動作は保証しません。

-delete

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-delete = <サブオプション>[,…]
          <サブオプション> : { [<ファイル>](<名前>[,…]) | <モジュール> }
```

- 省略時解釈

  なし

［詳細説明］

- 外部シンボル名またはライブラリモジュールを削除します。
- 特定のファイルに含まれるシンボル名，モジュールを削除することもできます。
- C/C++ 変数名，C 関数名はプログラム中での定義名先頭に _ を付加します。C++ 関数の場合は，引数列を含めたプログラム中の定義名をダブルクォーテーションで囲んで指定します。ただし，引数が void の場合は，" 関数名()" で指定します。同一ファイル名が複数存在する場合は，先に入力した方を優先します。
- 本オプションで，シンボル名削除を指定した場合，オブジェクトは削除されず，属性が内部シンボルに変更されます。

［例］

```
delete=(_sym1)            ; 全ファイル中のシンボル名 _sym1 を削除します。
delete=file1.obj(_sym2)   ; file1.obj 内のシンボル名 _sym2 を削除します。
```

［備考］

- extract または strip 指定時，本オプションは無効です。
- form=library のときに，モジュールを削除できます。
- form={absolute|relocate|hexadecimal|stype|binary} のときに，外部シンボルを削除できます。
- コンパイル・オプション -merge_files と本オプションを組み合わせて使用した場合，動作は保証しません。

-replace

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-replace = <サブオプション>[,…]
           <サブオプション> : <ファイル名>[(<モジュール名>[,…])]
```

- 省略時解釈

  なし

［詳細説明］

- ライブラリモジュールを置換します。
- 指定したファイルまたはライブラリモジュールと library オプションで指定したライブラリ内同名モジュールを置換します。

［例］

```
replace=file1.obj       ; モジュール file1 と file1.obj を置換します。
replace=lib1.lib(mdl1) ; モジュール mdl1 とライブラリファイル lib1.lib 内モジュール mdl1 を置換します。
```

［備考］

- form={object | relocate | absolute | hexadecimal | stype | binary} および extract，strip 指定時，本オプションは無効です。
- コンパイル・オプション -merge_files と本オプションを組み合わせて使用した場合，動作は保証しません。

-extract

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-extract = < モジュール名 >[,…]
```

- 省略時解釈

  なし

［詳細説明］

- ライブラリモジュールを抽出します。
- 指定したライブラリモジュールを library オプションで指定したライブラリファイルから抽出します。

［例］

```
extract=file1                ; モジュール file1 を抽出します。
```

［備考］

- form={absolute | hexadecimal | stype | binary} および strip 指定時，本オプションは無効です。
- form=library 指定時，モジュールを削除できます。
- form={absolute|relocate|hexadecimal|stype|binary} 指定時，外部シンボルを削除できます。

-strip

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-strip
```

- 省略時解釈

  なし

［詳細説明］

- アブソリュートファイル，ライブラリファイルのデバッグ情報を削除します。
- strip オプション指定時は，入力ファイルと出力ファイルは 1 対 1 対応になります。

［例］

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

- file1.abs, file2.abs, file3.abs のデバッグ情報を削除し，それぞれ file1.abs, file2.abs, file3.abs に出力します。デバッグ情報削除前のファイルは，file1.abk, file2.abk, file3.abk にバックアップします。

［備考］

- form={object | relocate | hexadecimal | stype | binary} 指定時，本オプションは無効です。

-change_message

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-change_message = ＜サブオプション＞[,…]
                  ＜サブオプション＞ : ＜エラーレベル＞[=＜エラー番号＞[-＜エラー番号＞][,…]]
                  ＜エラーレベル＞   : {Information | Warning | Error}
```

- 省略時解釈

  なし

［詳細説明］

- インフォメーション，ウォーニング，エラーレベルのメッセージレベルを変更します。
- メッセージ出力時の実行継続 / 中断を変更できます。
- エラー番号を指定すると，指定した番号のメッセージの種別を変更します。

  また，ハイフン（-）を使用して，メッセージ番号の範囲を指定することもできます。
- エラー番号には，コンポーネント番号（05），発生フェーズ（6）に続けて出力される 4 桁の数値（E0562310 の場合は 2310）を指定します。
- メッセージ番号の指定を省略した場合は，すべてのメッセージの種別を指定したものに変更します。

［例］

```
change_message=warning=2310
```

- E0562310 をウォーニングレベルに変更し、E0562310 出力時も処理を継続します。

```
change_message=error
```

- すべてのインフォメーション，ウォーニングメッセージをエラーレベルに変更します。
- メッセージを一つでも出力すると，処理を中断します。

［備考］

-

-hide

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-hide
```

- 省略時解釈

  なし

［詳細説明］

- 本オプションを指定した場合，出力ファイル内のローカルシンボル名情報を消去します。
- ローカルシンボルに関する名前の情報が消去されますので，バイナリエディタなどでファイルを開いてもローカルシンボル名は確認できなくなります。生成されるファイルの動作への影響は一切ありません。
- ローカルシンボル名を機密扱いにしたい場合などに本オプションを指定してください。
- 秘匿対象となるシンボルの種類を以下に挙げます。
  - ソースファイル：static 型修飾子を指定した変数名，関数名など
  - ソースファイル：goto 文のラベル名
  - アセンブリソース：外部定義（参照）シンボル宣言していないシンボル名

  ※ エントリ関数名は秘匿対象になりません。

［例］

- ソースファイルで本オプションの機能が有効となる記述の例を以下に示します。

```
int g1;
int g2=1;
const int g3=3;
static int s1;           //<--- static 変数名は秘匿対象
static int s2=1;         //<--- static 変数名は秘匿対象
static const int s3=2;   //<--- static 変数名は秘匿対象

static int sub1()        //<--- static 関数名は秘匿対象
{
    static int s1;       //<--- static 変数名は秘匿対象
    int l1;

    s1 = l1; l1 = s1;
    return(l1);
}

int main()
{
    sub1();
    if (g1==1)
        goto L1;
    g2=2;
L1:                      //<--- goto 文のラベル名は秘匿対象
    return(0);
}
```

［備考］

- 本オプションは出力ファイル形式が absolute,relocate,library の場合のみ有効です。
- コンパイル，アセンブル時に goptimize オプションを指定したファイルを入力する場合，出力ファイル形式が relocate,library の場合は本オプションを指定できません。
- 外部変数アクセス最適化を行う状況で本オプションを指定する場合は，一度目のリンク時には指定せず，二度目のリンク時にのみ本オプションを指定してください。
- デバッグ情報内のシンボル名は，本オプションを指定しても削除されません。

-total_size

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞

［指定形式］

```
-total_size
```

- 省略時解釈

  なし

［詳細説明］

- リンク後のセクションの合計サイズを，標準出力に表示するためのオプションです。
- 下記の 3 種類のセクションに分けて，合計サイズを表示します。
  - 実行可能なプログラムセクション
  - プログラムセクション以外の ROM 領域配置セクション
  - RAM 領域配置セクション
- 本オプションを使用することにより，ROM,RAM に配置する合計のセクションサイズを容易に認識することができます。

［備考］

- リンケージリストへ合計サイズを表示するには，別途 show=total_size オプションを使用する必要があります。
- ROM 化支援機能（rom オプション）対象のセクションの場合，転送元（ROM）と転送先（RAM）の両方で領域を使用するため，双方の合計サイズに対してセクションサイズを加算します。

サブコマンドファイルオプション

＜最適化リンケージエディタ（rlink）・オプション / サブコマンドファイルオプション＞

サブコマンドファイルオプションには，次のものがあります。

- -subcommand

-subcommand

＜最適化リンケージエディタ（rlink）・オプション / サブコマンドファイルオプション＞

［指定形式］

```
-subcommand = < ファイル名 >
```

- 省略時解釈

  なし

［詳細説明］

- オプションをサブコマンドファイルで指定します。
- サブコマンドファイルの書式は以下の通りです。

  < オプション > {= | △ } [< サブオプション > [,…]][△ &] [;< コメント >]
- オプションとサブオプションの区切りは，= の代わりに空白も指定できます。
- input オプションの場合は，サブオプション区切りに空白を指定できます。
- サブコマンドファイル内では 1 オプション / 行で指定します。
- サブオプションを 1 行に記述できない場合は，& を用いて継続指定できます。
- サブコマンドファイル中に subcommand オプションは指定できません。

［例］

- コマンドライン指定

```
rlink file1.obj -sub=test.sub file4.obj
```

- サブコマンド指定

```
input    file2.obj file3.obj      ; ここはコメントです。
library  lib1.lib, &              ; 継続行を指定します。
lib2.lib
```

- サブコマンドファイルで指定したオプション内容を，コマンドライン上のサブコマンド指定位置に展開し，実行します。
- ファイルの入力順序は，file1.obj, file2.obj, file3.obj, file4.obj になります。

残りのオプション

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞

残りのオプションには，次のものがあります。

- -logo
- -nologo
- -end
- -exit

-logo

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞

［指定形式］

```
-logo
```

- 省略時解釈

  コピーライト表示を出力します。

［詳細説明］

- コピーライト表示を出力します。

-nologo

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞

［指定形式］

```
-nologo
```

- 省略時解釈

  コピーライト表示を出力します。

［詳細説明］

- コピーライト表示出力を抑止します。

-end

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞

［指定形式］

```
-end
```

- 省略時解釈

  なし

［詳細説明］

- END より前に指定したオプション列を実行します。リンケージ処理終了後，END 以降に指定したオプション列の入力，リンケージ処理を継続します。
- 本オプションは，コマンドライン上では指定できません。

［例］

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

- (1) 〜 (3) の処理を実行し，a.abs を出力します。
- その後，(4) 〜 (6) の処理を実行し，a.mot を出力します。

-exit

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞

［指定形式］

```
-exit
```

- 省略時解釈

  なし

［詳細説明］

- オプション指定の終了を指定します。
- 本オプションは，コマンドライン上では指定できません。

［例］

- コマンドライン指定

```
rlink -sub=test.sub -nodebug
```

- test.sub

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

- (1) ～ (3) の処理を実行し，a.abs を出力します。
- Exit 実行後のコマンドライン指定の nodebug オプションは無効になります。

［備考］

-

(4) ライブラリジェネレータ・オプション

分類 .	オプション	説明
ライブラリオプション	-head	構築対象のライブラリを指定
	-output	出力ライブラリファイル名を指定
	-nofloat	簡易入出力関数の生成
	-lang	使用可能な C 言語標準ライブラリ関数構成の選択
	-simple_stdio	機能縮小版入出力関数の生成
	-logo -nologo	コピーライトを出力 コピーライトの出力を抑止

ライブラリオプション

<ライブラリジェネレータ・オプション / ライブラリオプション>

ライブラリオプションには，次のものがあります。

- -head
- -output
- -nofloat
- -lang
- -simple_stdio
- -logo
- -nologo

-head

<ライブラリジェネレータ・オプション / ライブラリオプション>

［指定形式］

```
-head=<sub>[,…]
<sub>:{ all | runtime | ctype | math | mathf | stdarg | stdio | stdlib | string | ios |
        new | complex | cppstring | c99_complex  |  fenv | inttypes | wchar | wctype}
```

- 省略時解釈

  head=all です。

［詳細説明］

- 構築対象をヘッダファイル名で指定します。
- head=all を指定した場合，すべてのヘッダファイル名が構築対象として指定されます。
- ランタイムライブラリは常に構築対象になります。

-output

<ライブラリジェネレータ・オプション / ライブラリオプション>

［指定形式］

```
-output=< ファイル名 >
```

- 省略時解釈

  output=stdlib.lib です。

［詳細説明］

- 出力ファイル名を指定します。

-nofloat

<ライブラリジェネレータ・オプション / ライブラリオプション>

[指定形式]

```
-nofloat
```

- 省略時解釈

  なし

[詳細説明]

- 浮動小数点変換（%f，%e，%E，%g，%G）をサポートしない，簡易入出力関数を生成します。
- 浮動小数点変換を必要としないファイル入出力を行う場合，ROM サイズを削減することができます。

  対象関数 fprintf，fscanf，printf，scanf，sprintf，sscanf，vfprintf，vprintf，vsprintf

[備考]

- 本オプションを指定して作成したライブラリでは，対象関数で浮動小数点数の入出力をした場合の動作は保証しません。

-lang

<ライブラリジェネレータ・オプション / ライブラリオプション>

［指定形式］

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

- 省略時解釈

  lang=c です。

［詳細説明］

- 使用可能な C 言語標準ライブラリ関数の構成を選択します。
- lang=c を選択すると，C 言語の標準関数を，C89 規格準拠のものだけで構成し，C99 規格で拡張された関数を含めません。lang=c99 を選択すると，C 言語の標準関数を，C89 規格および C99 規格準拠の内容で構成します。

［備考］

- C++,EC++ ライブラリの標準関数の構成は変化しません。
- lang=c99 を指定すると，C99 規格を含めたすべての関数が使用できますが，lang=c 指定時に比べて関数の数が多いため，ライブラリ生成に多くの時間が必要になる場合があります。

-simple_stdio

<ライブラリジェネレータ・オプション / ライブラリオプション>

[指定形式]

```
-simple_stdio
```

- 省略時解釈

  なし

[詳細説明]

- 機能縮小版の入出力関数を生成します。
- 機能縮小版では，浮動小数点の変換（nofloat オプションでサポートされない機能と同じ），long long 型の変換，2 バイトコードの変換が含まれません。これらの機能を必要としないファイル入出力を行う場合，ROM サイズを削減することができます。

  対象関数 fprintf，fscanf，printf，scanf，sprintf，sscanf，vfprintf，vprintf，vsprintf

[備考]

- 対象関数で縮小された機能を使用した場合，本オプションを指定して作成したライブラリをリンクした時の動作は保証しません。
- 本機能は，C++ または EC++ コンパイル時は無効です。

-logo

<ライブラリジェネレータ・オプション / ライブラリオプション>

[指定形式]

```
-logo
```

- 省略時解釈

  コピーライト表示が出力されます。

[詳細説明]

- コピーライト表示が出力されます。

-nologo

<ライブラリジェネレータ・オプション / ライブラリオプション>

[指定形式]

```
-nologo
```

- 省略時解釈

  コピーライト表示が出力されます。

[詳細説明]

- nologo オプション指定時は，コピーライトの表示の出力が抑止されます。

無効となるコンパイラオプション

ライブラリジェネレータでは、（4）ライブラリジェネレータ・オプションの他に C/C++ コンパイラオプションを指定し、ライブラリをコンパイルするときに用いるオプションとして選択することができます。ただし、以下に示すオプションは無効となり、ライブラリのコンパイルでは選択されません。

表 A—18　無効オプション一覧

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
1	lang	常に無効	なし
2	include	常に無効	なし
3	define	常に無効	なし
4	undefined	常に無効	なし
5	message nomessage	常に無効	nomessage
6	change_message	常に無効	なし
7	file_inline_path	常に無効	なし
8	comment	常に無効	なし
9	check	常に無効	なし
10	output	常に無効	output=obj
11	noline	常に無効	なし
12	debug nodebug	常に無効	nodebug
13	listfile nolistfile show	常に無効	nolistfile
14	file_inline	常に無効	なし
15	asmcmd	常に無効	なし
16	lnkcmd	常に無効	なし
17	asmopt	常に無効	なし
18	lnkopt	常に無効	なし
19	logo nologo	常に無効	nologo
20	euc sjis latin1 utf8	常に無効	なし
21	outcode	常に無効	なし
22	subcommand	常に無効	なし

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
23	alias	常に無効	alias=noansi
24	pic pid	lang=cpp または C++ ソースコンパイル時**注 1**	なし
25	ip_optimize	常に無効	なし
26	merge_files	常に無効	なし
27	whole_program	常に無効	なし
28	big5 gb2312	常に無効**注 2**	なし
29	map nomap	常に無効	smap

注 1. 警告 W0511171 が表示されます。

2. エラー F0593305 になります (ライブラリ生成できません)。

付録B 索 引

【O】

【P】

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.00	2013.10.01	―	初版発行

CC-RX V2.01.00　ユーザーズマニュアル
RX ビルド編

発行年月日　2013.10.01　　Rev.1.00

発行　　ルネサス エレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）　(03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

Colophon 1.1

CC-RX V2.01.00

RENESAS

ルネサス エレクトロニクス株式会社

R20UT2747JJ0100