User's Manual

CC-RH V1.02.00

RH850 ファミリ Cコンパイラパッケージ

ユーザーズマニュアル コンパイラ編

本資料に記載の全ての情報は発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

この資料に記載されている会社名，製品名などは，各社の商標または登録商標です

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2015.02

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。

この資料に記載されている会社名，製品名などは，各社の商標または登録商標です。

目次

1. 概　説 10
1.1 概　要 10
1.2 特　長 10
1.3 最 大 値 10
2. コマンド・リファレンス 11
2.1 概　要 11
2.2 入出力ファイル 12
2.3 環境変数 14
2.4 操作方法 15
2.4.1 コマンド・ラインでの操作方法 15
2.4.2 サブコマンド・ファイルの使用方法 17
2.5 オプション 18
2.5.1 コンパイル・オプション 19
2.5.2 アセンブル・オプション 101
2.5.3 リンク・オプション 132
2.6 オプションの複数指定 192
2.6.1 優先順序 192
2.6.2 機能矛盾 192
2.6.3 依存関係 192
2.6.4 #pragma 指定との関係 193
3. 出力ファイル 194
3.1 アセンブル・リスト・ファイル 194
3.1.1 アセンブル・リストの構成 194
3.1.2 アセンブル・リスト 194
3.1.3 セクション・リスト 195
3.1.4 コマンド・ライン情報 195
3.2 リンク・マップ・ファイル 196
3.2.1 リンク・マップの構成 196
3.2.2 オプション情報 196
3.2.3 エラー情報 197
3.2.4 リンク・マップ情報 197
3.2.5 合計セクション・サイズ 199
3.2.6 シンボル情報 199
3.2.7 クロス・リファレンス情報 202
3.3 リンク・マップ・ファイル（オブジェクト結合時） 203
3.3.1 リンク・マップの構成 203

3.3.2 ヘッダ情報 203
3.3.3 オプション情報. 203
3.3.4 エラー情報 204
3.3.5 エントリ情報 204
3.3.6 結合アドレス情報. 204
3.3.7 アドレス重複情報. 205
3.4 ライブラリ・リスト・ファイル. 206
3.4.1 ライブラリ・リストの構成. 206
3.4.2 オプション情報. 206
3.4.3 エラー情報 207
3.4.4 ライブラリ情報. 207
3.4.5 ライブラリ内モジュール，セクション，シンボル情報 207
3.5 インテル拡張ヘキサ・ファイル. 209
3.5.1 インテル拡張ヘキサ・ファイルの構成 209
3.5.2 スタート・リニア・アドレス・レコード. 210
3.5.3 拡張リニア・アドレス・レコード 210
3.5.4 スタート・アドレス・レコード 211
3.5.5 拡張アドレス・レコード 211
3.5.6 データ・レコード. 212
3.5.7 エンド・レコード. 212
3.6 モトローラ・S タイプ・ファイル 214
3.6.1 モトローラ・S タイプ・ファイルの構成 214
3.6.2 S0 レコード 215
3.6.3 S1 レコード 215
3.6.4 S2 レコード 215
3.6.5 S3 レコード 216
3.6.6 S7 レコード 216
3.6.7 S8 レコード 217
3.6.8 S9 レコード 217

4. コンパイラ言語仕様 218
4.1 基本言語仕様 218
4.1.1 未規定の項目 218
4.1.2 未定義の項目 218
4.1.3 処理系依存 220
4.1.4 C99 言語機能 228
4.1.5 ANSI 規格に厳密な処理オプション 230
4.1.6 データの内部表現と領域 232
4.1.7 セクション名 237
4.1.8 レジスタ・モード. 238
4.2 拡張言語仕様 240
4.2.1 予約語. 240

4.2.2 コンパイラ生成シンボル 240
4.2.3 #pragma 指令 240
4.2.4 拡張言語仕様の使用方法 242
4.2.4.1 データとプログラムのセクション割り当て 242
4.2.4.2 アセンブラ命令の記述. 248
4.2.4.3 インライン展開 249
4.2.4.4 割り込みレベルの制御. 251
4.2.4.5 割り込み／例外処理ハンドラ 252
4.2.4.6 マスカブル割り込みの禁止／許可 258
4.2.4.7 組み込み関数. 259
4.2.4.8 構造体パッキング 262
4.2.4.9 ビット・フィールドの割り付け 269
4.2.4.10 コア番号指定（マルチコア用）. 270
4.2.5 C ソースの修正. 272

5. アセンブラ言語仕様 274

5.1 ソースの記述方法 274
5.1.1 記述方法 274
5.1.2 式と演算子 280
5.1.3 算術演算子 282
5.1.4 論理演算子 290
5.1.5 比較演算子 295
5.1.6 シフト演算子 304
5.1.7 バイト分離演算子. 307
5.1.8 2 バイト分離演算子 310
5.1.9 セクション集合演算子 314
5.1.10 その他の演算子. 319
5.1.11 演算の制限 321
5.1.12 識 別 子. 322
5.2 疑似命令. 323
5.2.1 概　　要 323
5.2.2 セクション定義疑似命令 324
5.2.3 シンボル定義疑似命令 334
5.2.4 コンパイラ出力疑似命令 337
5.2.5 データ定義，領域確保疑似命令 343
5.2.6 外部定義，外部参照疑似命令. 354
5.2.7 マクロ疑似命令. 358
5.3 制御命令. 369
5.3.1 概　　要 369
5.3.2 アセンブラ制御命令 370
5.3.3 ファイル入力制御命令 378
5.3.4 条件アセンブル制御命令 381

5.4 マ ク ロ 390
5.4.1 概　　要 390
5.4.2 マクロの利用 390
5.4.3 マクロ・オペレータ 390
5.5 予 約 語 392
5.6 あらかじめ定義されたマクロ名 392
5.7 アセンブラ生成シンボル 392
5.8 命令セット 394
5.9 命令の説明 403
5.9.1 ロード／ストア命令 405
5.9.2 算術演算命令 417
5.9.3 飽和演算命令 469
5.9.4 論理演算命令 478
5.9.5 分岐命令 513
5.9.6 ビット操作命令 530
5.9.7 スタック操作命令 539
5.9.8 特殊命令 544
5.9.9 ループ命令 577
5.9.10 浮動小数点演算命令 580
5.9.11 その他の命令 654

6. セクション仕様 655
6.1 セクション 655
6.1.1 セクションの結合 655
6.2 特殊シンボル 657

7. ライブラリ関数仕様 659
7.1 提供ライブラリ 659
7.2 ヘッダ・ファイル 659
7.3 リエントラント性 660
7.4 ライブラリ関数 661
7.4.1 プログラム診断機能関数 661
7.4.2 可変個引数関数 663
7.4.3 文字列関数 667
7.4.4 メモリ管理関数 683
7.4.5 文字変換関数 689
7.4.6 文字分類関数 692
7.4.7 標準入出力関数 705
7.4.8 標準ユーティリティ関数 737
7.4.9 非局所分岐関数 765
7.4.10 数学関数 768
7.4.11 RAM セクション領域初期化関数 814
7.4.12 周辺装置の初期化関数 816

7.4.13 演算用ランタイム関数 818

8. スタートアップ 820

8.1 概　　要 820

8.2 スタートアップ・ルーチン 820

8.2.1 例外ベクタ・テーブル 820

8.2.2 ブート・ローダ・プロジェクト用スタートアップ・ルーチン 821

8.2.3 アプリケーション・プロジェクト用スタートアップ・ルーチン 823

8.2.4 アプリケーション・プロジェクトからブート・ローダ・プロジェクトへの情報の受け渡し 827

8.3 コーディング例 828

8.4 シンボル 840

8.4.1 グローバル・ポインタ（gp） 840

8.4.2 エレメント・ポインタ（ep） 842

8.5 ROM 化 846

8.5.1 概　　要 846

8.5.2 ROM 化用ロード・モジュールの作成 846

9. 関数呼び出し仕様 848

9.1 関数呼び出しインタフェース 848

9.1.1 関数呼び出し前後で保証される汎用レジスタ 848

9.1.2 引数と戻り値の設定方法，および参照方法 848

9.1.3 スタック・ポインタが指すアドレス 850

9.1.4 スタック・フレーム 850

9.2 C 言語からアセンブリ言語ルーチンの呼び出し 853

9.3 アセンブリ言語から C 言語ルーチンの呼び出し 854

9.4 他言語で定義された変数の参照 854

9.5 汎用レジスタ 855

10. メッセージ 856

10.1 概　　説 856

10.2 出力形式 856

10.3 メッセージ種別 856

10.4 メッセージ番号 856

10.5 メッセージ 856

10.5.1 内部エラー 857

10.5.2 フェイタル・エラー 859

10.5.3 アボート・エラー 943

10.5.4 インフォメーション 963

10.5.5 ワーニング 967

11. 注意事項 1009

11.1 volatile 修飾子 1009

11.2 アセンブラにおける V850E3v5 G3K コア指定（-Xcpu=g3k オプション） 1010

A. クイック・ガイド 1011
A.1 変数（C 言語） 1011
A.1.1 短い命令長でアクセスできる領域へ配置する 1011
A.1.1.1 GP 相対アクセス 1011
A.1.1.2 EP 相対アクセス 1011
A.1.2 配置領域を変更する 1012
A.1.3 -Xpreinclude オプションを使用して配置領域を変更する 1014
A.1.4 通常時と割り込み時に使用する変数を定義する 1014
A.1.5 const 定数ポインタを定義する 1015
A.2 関 数 1016
A.2.1 配置領域を変更する 1016
A.2.2 離れた関数をコールする 1016
A.2.3 アセンブラ命令の埋め込み 1017
A.2.4 RAM で実行する 1017
A.3 変数（アセンブラ） 1017
A.3.1 初期値なし変数を定義する 1017
A.3.2 初期値あり変数を定義する 1018
A.3.3 const 定数を定義する 1018

1. 概　説

この章では，RH850 ファミリ用 C コンパイラ・パッケージ（CC-RH）の全体概要について説明します。

1.1 概　要

CC-RH は，C 言語，またはアセンブリ言語で記述されたプログラムを機械語に変換するプログラムです。

1.2 特　長

CC-RH は，次の特長を備えています。

(1) ANSI 規格に準拠した言語仕様
C 言語仕様は，ANSI 規格に準拠しています。

(2) 高度な最適化
高度な最適化を適用し，従来の最適化に加えて大域最適化を実現します。
これにより，コード・サイズ，および実行速度に優れたコードを生成し，かつコンパイル時間も短縮されます。

(3) 高い移植性
既存コンパイラ（SuperH RISC engine C/C++ コンパイラ）向けプログラムからの移行をサポートします。
また，デバッグ情報には業界標準フォーマットである DWARF2 を採用しています。

(4) 多機能性
CS+ との連携による静的解析機能などを提供します。

1.3 最 大 値

C 言語，またはアセンブリ言語でプログラムを記述する際に留意すべき最大値を以下に示します。

(1) コンパイラの最大値
翻訳に際しての限界値はありません。翻訳可能な最大値は動作するホスト・マシンのメモリに依存します。
ただし，アセンブラが規定する最大値を越える場合は，エラーとなります。

備考　1 つのオブジェクトのバイト数の最大値は 2G バイトとなります（ホスト環境の場合）。

(2) アセンブラの最大値
アセンブリ言語でプログラムを記述する際に留意すべき最大値を以下に示します。

表 1.1　アセンブラの最大値

項目	最大値
シンボルの長さ（トークンの長さ）	4,294,967,294 注
ラベルの長さ（トークンの長さ）	4,294,967,294 注
シンボルの最大個数	4,294,967,294 注
LOCAL 疑似命令の仮パラメータ数	4,294,967,294 注
LOCAL 疑似命令の自動生成シンボル数	4,294,967,294 注
INCLUDE 制御命令のネスト	4,294,967,294 注
TDATA 再配置属性セクション	256 バイト
ALIGN 疑似命令	2 以上 2^{31} 未満の偶数
IRP 疑似命令の実パラメータ	4,294,967,294 注
LOCAL 疑似命令の仮パラメータ	4,294,967,294 注

注　動作するホスト・マシンのメモリに依存します。

2. コマンド・リファレンス

ここでは，ビルド・ツール（CC-RH）に含まれる各コマンドの仕様についての詳細を説明します。

2.1 概　　要

CC-RH は，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。

以下のコマンドで構成されており，1 つのコンパイル・ドライバ（ccrh）がコンパイルからリンクまでの全フェーズを制御します。

ccrh　：コンパイル・ドライバの起動コマンド
asrh　：アセンブラの起動コマンド
rlink　：最適化リンカの起動コマンド

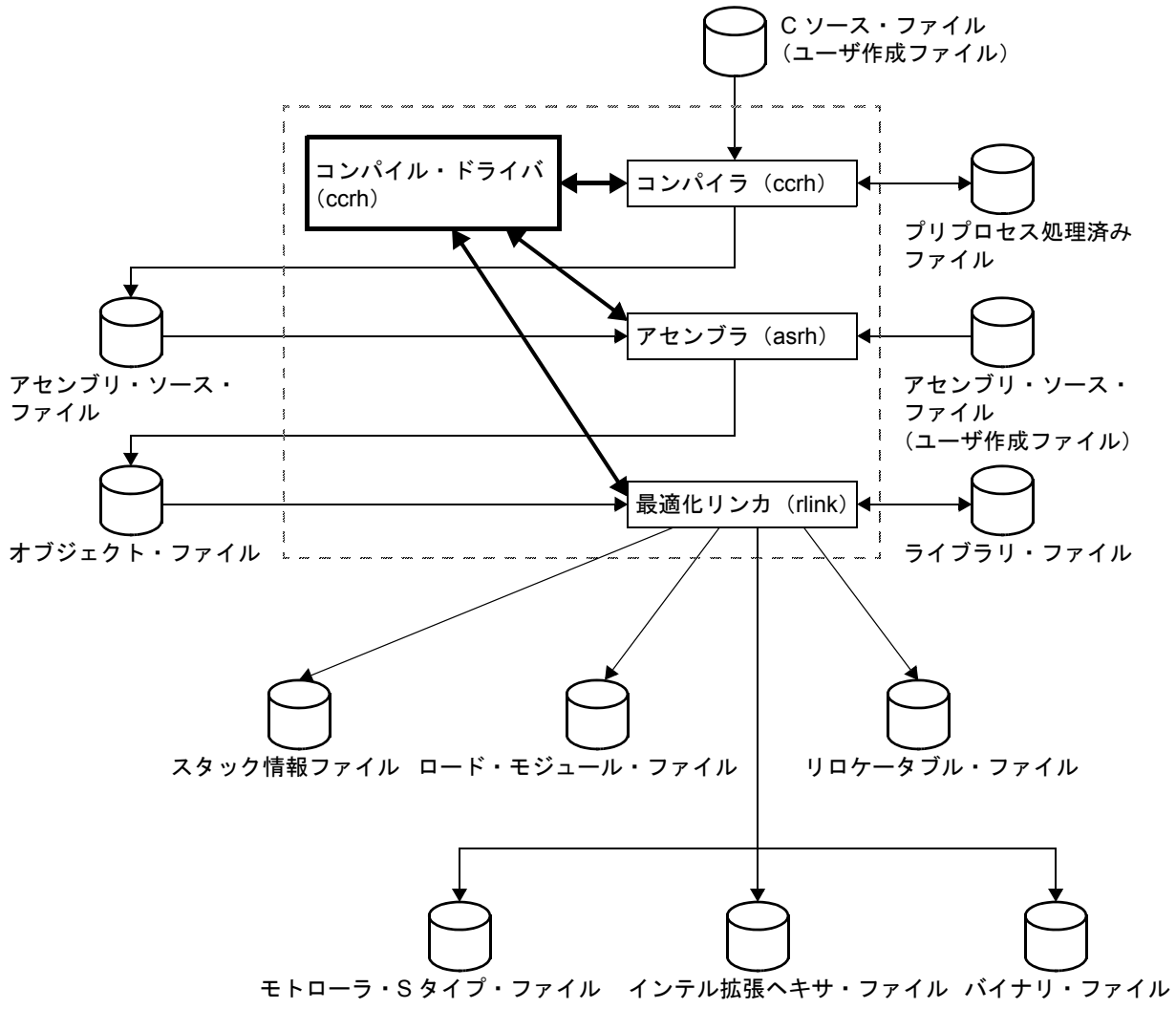
各コマンドの処理について説明します。

(1) コンパイラ（ccrh）
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，アセンブリ・ソース・プログラムを生成します。

(2) アセンブラ（asrh）
アセンブリ・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・ファイルを生成します。

(3) 最適化リンカ（rlink）
オブジェクト・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。
また，ROM 化処理，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，インテル拡張ヘキサ・ファイルやモトローラ・S タイプ・ファイルへの変換を行います。

図 2.1 ccrh における処理の流れ



2.2 入出力ファイル

コンパイル・ドライバである ccrh コマンドの入出力ファイルを以下に示します。

表 2.1 ccrh コマンドの入出力ファイル

ファイル種別	拡張子	入出力	説明
C ソース・ファイル	.c	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。
プリプロセス処理済みファイル	.i 注 1	出力	入力ファイル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・ファイルです。 -P オプション指定時に出力します。
アセンブリ・ソース・ファイル	.asm 注 1	出力	コンパイルにより C ソースから生成したアセンブリ言語ファイル -S オプション指定時に出力します。
	.asm .s	入力	アセンブリ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。

ファイル種別	拡張子	入出力	説明
ヘッダ・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，また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。 拡張子は任意ですが，以下を推奨します。 - #include 指令：.h - $include 制御命令：.inc
オブジェクト・ファイル	.obj 注 1	入出力	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
アセンブル・リスト・ファイル注 2	.prn 注 1	出力	アセンブル結果の情報を持つリスト・ファイル -Xasm_option=-Xprn_path オプション指定時に出力します。
ライブラリ・ファイル	.lib 注 1	入出力	複数のオブジェクト・ファイルが登録された ELF 形式ファイル -Xlk_option=-form=library オプション指定時に出力します。
ロード・モジュール・ファイル	.abs 注 1	入出力	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。 -Xlk_option=-form=absolute オプション指定時に出力します。 ただし，-Xlk_option オプションで -form オプションが指定されていない場合は，上記オプションが指定されたものと解釈して処理します。
リロケータブル・ファイル	.rel 注 1	出力	リロケータブルなオブジェクト・ファイル -Xlk_option=-form=relocate オプション指定時に出力します。
インテル拡張ヘキサ・ファイル注 2	.hex 注 1	入出力	ロード・モジュール・ファイルをインテル拡張ヘキサ・フォーマットに変換したファイル -Xlk_option=-form=hexadecimal オプション指定時に出力します。
モトローラ・S タイプ・ファイル注 2	.mot 注 1	入出力	ロード・モジュール・ファイルをモトローラ・S タイプに変換したファイル -Xlk_option=-form=stype オプション指定時に出力します。
バイナリ・ファイル	.bin 注 1	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 -Xlk_option=-form=binary 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	.fsy	入出力	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リ・ソース・ファイル -Xlk_option=-fsymbol オプション指定時に出力します。
リンク・マップ・ファイル注 2	.map 注 1	出力	リンク結果の情報を持つリスト・ファイル -Xlk_option=-list オプション指定時に出力します。
ライブラリ・リスト・ファイル注 2	.lbp 注 1	出力	ライブラリ作成結果の情報を持つリスト・ファイル -Xlk_option=-list オプション指定時に出力します。
スタック情報ファイル	.sni	出力	スタック使用量の情報を持つファイル -Xlk_option=-stack オプション指定時に出力します。

ファイル種別	拡張子	入出力	説明
外部シンボル割り付け情報ファイル	.bls 注 1	出力	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 -Omap オプション指定時に出力します。
静的解析情報ファイル	任意	入出力	本製品が利用する情報を持つファイル 拡張子は任意ですが，“.cref” を推奨します。 -Xcref オプション指定時に出力します。
エラー・メッセージ・ファイル	任意	出力	エラー・メッセージを内容とするファイル 拡張子は任意ですが，“.err” を推奨します。 -Xerror_file オプション指定時に出力します。
サブコマンド・ファイル	任意	入力	実行プログラムのパラメータを内容とするファイル ユーザ作成ファイルです。

注 1. オプション指定により拡張子の変更が可能です。

注 2. 各ファイルについての詳細は，「3. 出力ファイル」を参照してください。

2.3 環境変数

ここでは，環境変数について説明します。
最適化リンカの環境変数とコマンド・ラインでの指定例を以下に示します。

- HLNK_LIBRARY1，HLNK_LIBRARY2，HLNK_LIBRARY3
  最適化リンカが使用するデフォルト・ライブラリ名を指定します。
  ライブラリは，-library オプションで指定したもの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，HLNK_LIBRARY1，HLNK_LIBRARY2，HLNK_LIBRARY3 の順に，デフォルト・ライブラリを検索します。

  例

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

- HLNK_TMP
  最適化リンカがテンポラリ・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・フォルダに作成します。

  例

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

- HLNK_DIR
  最適化リンカ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 指定フォルダとなります。
  ただし，ワイルドカードで指定したファイルは，カレント・フォルダ内だけを検索します。

  例

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

2.4 操作方法

ここでは，各コマンドの操作方法について説明します。

- コマンド・ラインでの操作方法
- サブコマンド・ファイルの使用方法

2.4.1 コマンド・ラインでの操作方法

コンパイル・ドライバである ccrh コマンドの起動により，コンパイル，アセンブル，リンクなどを行うことができます。
また，アセンブラ（asrh），最適化リンカ（rlink）は単体起動することも可能です。

(1) 指定形式
各コマンドは，コマンド・ラインで以下のように入力します。

```
>ccrh[ △ option]... △ file[ △ file| △ option]...
```

```
>asrh[ △ option]... △ file[ △ file| △ option]...
```

```
>rlink[{ △ file| △ option}...]
```

option ： オプション名
file ： ファイル名
[ ] ： [ ] 内は省略可能
... ： 直前の [ ] 内のパターンの繰り返しが可能
{ } ： | で区切られた項目を選択
△ ： 1 個以上の空白

コマンドを入力する際の注意事項を以下に示します。

- オプションの指定形式は，使用するコマンドによって異なります。
  各コマンドのオプションに対する注意事項については，「2.5.1 コンパイル・オプション」，「2.5.2 アセンブル・オプション」，「2.5.3 リンク・オプション」を参照してください。
- ファイル名は，Windows で認められるものであれば指定可能です。
  ただし，“@” はサブコマンド・ファイル指定と判断するため，“@” をファイル名の先頭文字として使用することはできません。
  また，“-” はオプション指定と判断するため，“-” もファイル名の先頭文字として使用することはできません。
  さらに，“(”，“)” はリンク・オプションの一部と判断するため，“(”，“)” をファイル名に使用することはできません。
- 指定可能なファイル名の長さは，Windows に依存します（259 文字まで）。
- ファイル名のアルファベットは，大文字／小文字を区別しません。
- 入力として複数のファイルが指定可能です。
  異なる種類のファイル（C ソース・ファイルとアセンブリ・ソース・ファイル，またはオブジェクト・ファイルなど）を混在して指定することも可能です。
  ただし，拡張子を除いたソース・ファイル名が同じファイルを複数指定することはできません（異なるフォルダにある場合も含みます）。
  複数のファイルを指定した場合は，1 つのファイルでエラーとなっても，残りのファイルの処理が継続可能であれば，処理を継続します。
  なお，生成したオブジェクト・ファイルはリンク後も削除しません。

(2) 操作例
コマンド・ラインでの操作例を以下に示します。

備考 各オプションについての詳細は，「2.5 オプション」を参照してください。

(a) コンパイル，アセンブル，リンクを 1 コマンドで行う場合
C ソース・ファイル file1.c を ccrh でコンパイルして，アセンブリ・ソース・ファイル file1.asm を生成します。
次に，アセンブリ・ソース・ファイル file1.asm と file2.asm を asrh でアセンブルして，オブジェクト・ファイル file1.obj，file2.obj を生成します。
また，アセンブル・リスト・ファイル sample.prn も出力します。
最後に，オブジェクト・ファイル file1.obj，file2.obj，file3.obj を rlink でリンクして，リンク・マップ・ファイル sample.map，およびロード・モジュール・ファイル sample.abs を生成します。

```
>ccrh file1.c file2.asm file3.obj -Xasm_option=-Xprn_path=sample.prn -
Xlk_option=-list -osample.abs -Xcommon=v850e3v5
```

備考 ccrh に対して，asrh のみに指定可能なオプションを指定する場合は -Xasm_option オプション，rlink のみに指定可能なオプションを指定する場合は -Xlk_option オプションを使用します。

(b) コンパイル，アセンブルを 1 コマンドで行い，リンクは単独で行う場合
C ソース・ファイル file1.c を ccrh でコンパイルして，アセンブリ・ソース・ファイル file1.asm を生成します。
次に，アセンブリ・ソース・ファイル file1.asm と file2.asm を asrh でアセンブルして，オブジェクト・ファイル file1.obj，file2.obj を生成します。
また，アセンブル・リスト・ファイル sample.prn も出力します。

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

備考 ccrh に対して，asrh のみに指定可能なオプションを指定する場合は -Xasm_option オプションを使用します。

オブジェクト・ファイル file1.obj，file2.obj，file3.obj を rlink でリンクして，リンク・マップ・ファイル sample.map，およびロード・モジュール・ファイル sample.abs を生成します。

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

(c) コンパイル，アセンブル，リンクともに単独で行う場合
C ソース・ファイル file1.c を ccrh でコンパイルして，アセンブリ・ソース・ファイル file1.asm を生成します。

```
>ccrh -S file1.c -Xcommon=v850e3v5
```

アセンブリ・ソース・ファイル file1.asm と file2.asm を asrh でアセンブルして，オブジェクト・ファイル file1.obj，file2.obj を生成します。
また，アセンブル・リスト・ファイル sample.prn も出力します。

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

オブジェクト・ファイル file1.obj，file2.obj，file3.obj を rlink でリンクして，リンク・マップ・ファイル sample.map，およびロード・モジュール・ファイル sample.abs を生成します。

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

2.4.2 サブコマンド・ファイルの使用方法

サブコマンド・ファイルとは，コマンドに対して指定するオプションやファイル名を記述したファイルです。
コマンドは，サブコマンド・ファイルの内容をコマンド・ラインの引数のように扱います。
サブコマンド・ファイルは，コマンド・ラインで引数を指定しきれない場合や，コマンドを実行するたびに同じオプションを繰り返し指定するような場合に使用します。

(1) コンパイラ，アセンブラでサブコマンド・ファイルを使用する場合

(a) サブコマンド・ファイルの記述に関する注意事項

- 指定する引数は，複数行に分けて記述することができます。
  ただし，オプションやファイル名の途中で改行することはできません。
- サブコマンド・ファイルをネストすることはできません。
- サブコマンド・ファイルの文字コードは，-Xcharacter_set オプションで指定することはできません。
  サブコマンド・ファイル中に ASCII 文字以外を使用する場合は，BOM 付き UTF-8，または SJIS のファイルにしてください。
- 以下の文字は，特殊文字として扱います。
  これらの特殊文字自体は，ccrh コマンドへのコマンド・ラインの中に含まずに削除します。

"（ダブルクォーテーション）	次のダブルクォーテーションまでの文字列を連続した文字列として扱います。
#（シャープ）	行頭に指定した場合は，行末までをコメントとして扱います。
^（ハット）	直後の文字を特殊文字として扱いません。

(b) サブコマンド・ファイルの指定例
サブコマンド・ファイル sub.txt をエディタで作成します。

```
-Xcommon=v850e3v5
-c
-Dtest
-Idir
-Osize
```

コマンド・ラインにおいて，サブコマンド・ファイル指定オプション @ により sub.txt を指定します。

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

コマンド・ラインは以下のように展開されます。

```
>ccrh -Xcommon=v850e3v5 -c -Dtest -Idir -Osize -ofile.obj file.c
```

(2) 最適化リンカでサブコマンド・ファイルを使用する場合

(a) サブコマンド・ファイルの記述に関する注意事項

- オプションは，先頭のハイフン（-）を省略して指定することができます。
- オプションとパラメータの区切りは，“=” の代わりに空白を使用することも可能です。
- 1 行につき 1 つのオプションを指定します。
  1 行に記述できない場合は，“&” を使用して複数行に記述することができます。
- サブコマンド・ファイル内に -subcommand オプションを指定することはできません。
- 以下の文字は，特殊文字として扱います。
  これらの特殊文字自体は，rlink コマンドへのコマンド・ラインの中に含まずに削除します。

&（アンド）	次の行を継続行として扱います。
;（セミコロン）	行末までをコメントとして扱います。

(b) サブコマンド・ファイルの指定例
サブコマンド・ファイル sub.txt をエディタで作成します。

```
input file2.obj file3.obj        ; ここはコメントです。
library lib1.lib, &              ;“&” は継続行を示します。
lib2.lib
```

コマンド・ラインにおいて，サブコマンド・ファイル指定オプション -subcommand により sub.txt を指定します。

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

コマンド・ラインは以下のように展開されます。

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

2.5 オプション

ここでは，ccrh 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→「2.5.1 コンパイル・オプション」参照
アセンブル・フェーズ→「2.5.2 アセンブル・オプション」参照
リンク・フェーズ→「2.5.3 リンク・オプション」参照

2.5.1 コンパイル・オプション

ここでは，コンパイル・フェーズのオプションについて説明します。

オプションに関する注意事項を以下に示します。

- オプションの大文字／小文字は区別します。
- パラメータとして数値を指定する場合は，10 進数，または “0x”（“0X”）で始まる 16 進数での指定が可能です。16 進数のアルファベットは，大文字／小文字を区別しません。,
- パラメータとしてファイル名を指定する場合は，パス付き（絶対パス，または相対パス）での指定が可能です。パスなし，および相対パスで指定する場合は，カレント・フォルダを基準とします。
- パラメータ中に空白を含める場合（パス名など）は，そのパラメータ全体をダブルクォーテーション（"）で囲んでください。

オプションの分類と説明を以下に示します。

表 2.2 コンパイル・オプション

分類	オプション	説明
バージョン／ヘルプ表示指定	-V	ccrh のバージョン情報を表示します。
	-h	ccrh のオプションの説明を表示します。
出力ファイル指定	-o	出力ファイル名を指定します。
	-Xobj_path	コンパイル途中に生成されるオブジェクト・ファイルを保存するフォルダを指定します。
	-Xasm_path	コンパイル途中に生成されるアセンブリ・ソース・ファイルを保存するフォルダを指定します。
	-Xprep_path	プリプロセス処理済みファイルを保存するフォルダを指定します。
ソース・デバッグ制御	-g	ソース・デバッグ用の情報を出力します。
デバイス指定	-Xcommon	デバイス共通のオブジェクト・ファイルを生成することを指定します。
	-Xcpu	指定したコア向けのオブジェクトを生成することを指定します。
処理中断指定	-P	入力ファイルに対してプリプロセス処理のみ実行します。
	-S	アセンブル以降の処理を実行しません。
	-c	リンク以降の処理を実行しません。
プリプロセッサ制御	-D	プリプロセッサ・マクロ，およびアセンブラ・シンボルを定義します。
	-U	-D オプションによるプリプロセッサ・マクロ，およびアセンブラ・シンボルの定義を解除します。
	-I	インクルード・ファイルを検索するフォルダを指定します。
	-Xpreinclude	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。
	-Xpreprocess	プリプロセス結果の出力を制御します。

分類	オプション	説明
C 言語制御	-Xansi	C ソース・プログラムを ANSI 規格に厳密にあわせて処理します。
	-Xenum_type	列挙型に対して，どの整数型として扱うかを指定します。
	-Xvolatile	外部変数の volatile 化を指定します。
	-Xcheck	C ソース・ファイルの互換性をチェックします。
	-Xmisra2004	MISRA-C:2004 ルールによるソース・チェックを行います。
	-Xignore_files_misra	MISRA-C:2004 ルールによるソース・チェックの対象外のファイルを指定します。
	-Xcheck_language_extention	言語拡張により部分抑止される MISRA-C:2004 ルールのソース・チェックを有効にします。
日本語／中国語文字列制御	-Xcharacter_set	日本語／中国語の文字コードを指定します。
最適化指定	-O	最適化のレベル，または各最適化項目の詳細を指定します。
	-Xintermodule	大域最適化を行います。
	-Xinline_strcpy	標準ライブラリ関数 strcpy，strcmp，memcpy，memset の呼び出しをインライン展開します。
	-Xmerge_string	文字列定数をマージします。
	-Xalias	ポインタ指示先の型を考慮した最適化の指定を行います。
	-Xmerge_files	複数の C ソース・ファイルをマージしてコンパイルを行います。
	-Xwhole_program	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て最適化を行います。

分類	オプション	説明
生成コード制御	-Xpack	構造体パッキングを行います。
	-Xbit_order	ビット・フィールドのメンバの並び順を指定します。
	-Xpass_source	出力するアセンブリ・ソース・ファイル中に C ソース・プログラムをコメントとして出力します。
	-Xswitch	switch 文のコード出力方式を指定します。
	-Xreg_mode	レジスタ・モードを指定します。
	-Xreserve_r2	r2 レジスタを予約します。
	-Xep	ep レジスタの扱い方を指定します。
	-Xfloat	浮動小数点演算命令の生成を制御します。
	-Xcall_jump	関数呼び出しの分岐命令の生成を制御します。
	-Xfar_jump	far jump の出力を制御します。
	-Xdiv	除算に対して，div，および divu 命令を生成します。
	-Xcheck_div_ov	除算時に OV フラグのチェックを行います。
	-Xuse_fmaf	積和演算命令を生成します。
	-Xunordered_cmpf	浮動小数点比較において，無効演算例外の検出を行います。
	-Xmulti_level	マルチコア用プログラムの生成を指定します。
	-Xpatch	パッチを適用します。
	-Xdbl_size	double 型および long double 型の精度を指定します。
	-Xround	浮動小数点定数の丸めモードを指定します。
	-Xalign4	関数の先頭アドレスのアライメントを指定します。
	-Xsection	データのデフォルトセクションを指定します。
情報ファイル出力制御	-Xcref	静的解析情報ファイルを出力します。
エラー出力制御	-Xerror_file	エラー・メッセージをファイルに出力します。
警告メッセージ出力制御	-Xno_warning	指定した警告メッセージの出力を抑止します。
フェーズ個別オプション指定	-Xasm_option	アセンブル・オプションを指定します。
	-Xlk_option	リンク・オプションを指定します。
サブコマンド・ファイル指定	@	サブコマンド・ファイルを指定します。

バージョン／ヘルプ表示指定

バージョン／ヘルプ表示指定オプションには，次のものがあります。

- -V
- -h

-V

ccrh のバージョン情報を表示します。

[指定形式]

```
-V
```

- 省略時解釈
  ccrh のバージョン情報を表示せずに，コンパイルを行います。

[詳細説明]

- ccrh のバージョン情報を標準エラー出力に出力します。
  コンパイルは行いません。

[使用例]

- ccrh のバージョン情報を標準エラー出力に出力します。

```
>ccrh -V -Xcommon=v850e3v5
```

-h

ccrh のオプションの説明を表示します。

[指定形式]

```
-h
```

- 省略時解釈
  ccrh のオプションの説明を表示しません。

[詳細説明]

- ccrh のオプションの説明を標準エラー出力に出力します。
  コンパイルは行いません。

[使用例]

- ccrh のオプションの説明を標準エラー出力に出力します。

```
>ccrh -h -Xcommon=v850e3v5
```

出力ファイル指定

出力ファイル指定オプションには，次のものがあります。

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

-o

出力ファイル名を指定します。

[指定形式]

```
-ofile
```

- 省略時解釈
  出力ファイル名は，指定オプションにより異なります。
  なお，ファイルの出力先はカレント・フォルダとなります。
  - -P オプションを指定している場合
    出力ファイル名は，入力ファイル名の拡張子を “.i” に置き換えたものとなります。
  - -S オプションを指定している場合
    出力アセンブリ・ソース・ファイル名は，ソース・ファイル名の拡張子を “.asm” に置き換えたものとなります。
  - -c オプションを指定している場合
    出力オブジェクト・ファイル名は，ソース・ファイル名の拡張子を “.obj” に置き換えたものとなります。
  - 上記以外の場合
    出力ロード・モジュール・ファイル名は，最初に入力したファイル名の拡張子を “.abs” に置き換えたものとなります。

[詳細説明]

- 出力ファイル名を *file* に指定します。
- *file* がすでに存在する場合は，そのファイルを上書きします。
- -P/-S/-c オプションと同時に指定することにより処理を中断した場合にも，本オプションは有効となります。
  - -P オプションと同時に指定した場合
    file には，入力ファイルに対してプリプロセス処理を行った結果のファイル名を指定したものとみなします。
  - -S オプションと同時に指定した場合
    file には，アセンブリ・ソース・ファイル名を指定したものとみなします。
  - -c オプションと同時に指定した場合
    file には，オブジェクト・ファイル名を指定したものとみなします。
  - 上記以外の場合
    file には，ロード・モジュール・ファイル名を指定したものとみなします。
- 出力ファイルが複数の場合は，エラーとなります。
- *file* を省略した場合は，エラーとなります。

[使用例]

- ロード・モジュール・ファイルをファイル名 sample.abs で出力します。

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

-Xobj_path

コンパイル途中に生成されるオブジェクト・ファイルを保存するフォルダを指定します。

[指定形式]

```
-Xobj_path[=path]
```

- 省略時解釈
  カレント・フォルダに，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。

[詳細説明]

- コンパイル途中に生成されるオブジェクト・ファイルを保存するフォルダを *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。
  存在しないフォルダ名を指定した場合は，エラーとなります。
- *path* には存在するファイル名を指定することも可能です。
  出力するオブジェクト・ファイルが 1 つの場合は，*path* というファイル名で保存します。
  出力するオブジェクト・ファイルが複数の場合は，エラーとなります。
  存在しないファイル名を指定した場合は，エラーとなります。
- =*path* を省略した場合は，カレント・フォルダに，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。
- ソース・ファイルとして同じ名前のファイル（異なるフォルダにある場合を含む）を複数指定した場合は，警告を出力して，最後に指定したソース・ファイルに対するオブジェクト・ファイルのみを保存します。

[使用例]

- コンパイル途中に生成されるオブジェクト・ファイルをフォルダ D:¥sample に保存します。

```
>ccrh -Xobj_path=D:¥sample -Xcommon=v850e3v5 main.c
```

-Xasm_path

コンパイル途中に生成されるアセンブリ・ソース・ファイルを保存するフォルダを指定します。

[指定形式]

```
-Xasm_path[=path]
```

- 省略時解釈
  アセンブリ・ソース・ファイルを出力しません（-S オプション指定時を除く）。

[詳細説明]

- コンパイル途中に生成されるアセンブリ・ソース・ファイルを保存するフォルダを *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，C ソース・ファイル名の拡張子を “.asm” で置き換えたファイル名でアセンブリ・ソース・ファイルを保存します。
  存在しないフォルダ名を指定した場合は，エラーとなります。
- *path* には存在するファイル名を指定することも可能です。
  出力するアセンブリ・ソース・ファイルが 1 つの場合は，*path* というファイル名で保存します。
  出力するアセンブリ・ソース・ファイルが複数の場合は，エラーとなります。
  存在しないファイル名を指定した場合は，エラーとなります。
- =*path* を省略した場合は，カレント・フォルダに，C ソース・ファイル名の拡張子を “.asm” で置き換えたファイル名でアセンブリ・ソース・ファイルを保存します。
- ソース・ファイルとして同じ名前のファイル（異なるフォルダにある場合を含む）を複数指定した場合は，警告を出力して，最後に指定したソース・ファイルに対するアセンブリ・ソース・ファイルのみを保存します。

[使用例]

- コンパイル途中に生成されるアセンブリ・ソース・ファイルをフォルダ D:¥sample に保存します。

```
>ccrh -Xasm_path=D:¥sample -Xcommon=v850e3v5 main.c
```

-Xprep_path

プリプロセス処理済みファイルを保存するフォルダを指定します。

[指定形式]

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

- 省略時解釈
  プリプロセス処理済みファイルを出力しません（-P オプション指定時を除く）。

[詳細説明]

- プリプロセス処理済みファイルを保存するフォルダを *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，C ソース・ファイル名の拡張子を ".i" で置き換えたファイル名でプリプロセス処理済みファイルを保存します。
  存在しないフォルダ名を指定した場合は，エラーとなります。
- *path* には存在するファイル名を指定することも可能です。
  出力するプリプロセス処理済みファイルが 1 つの場合は，*path* というファイル名で保存します。
  出力するプリプロセス処理済みファイルが複数の場合は，エラーとなります。
  存在しないファイル名を指定した場合は，エラーとなります。
- =*path* を省略した場合は，カレント・フォルダに，C ソース・ファイル名の拡張子を ".i" で置き換えたファイル名でプリプロセス処理済みファイルを保存します。
- ソース・ファイルとして同じ名前のファイル（異なるフォルダにある場合を含む）を複数指定した場合は，警告を出力して，最後に指定したソース・ファイルに対するプリプロセス処理済みファイルのみを保存します。

[使用例]

- プリプロセス処理済みファイルをフォルダ D:¥sample に保存します。

```
>ccrh -Xprep_path=D:¥sample -Xcommon=v850e3v5 main.c
```

ソース・デバッグ制御

ソース・デバッグ制御オプションには，次のものがあります。

- -g

-g

ソース・デバッグ用の情報を出力します。

[指定形式]

```
-g
```

- 省略時解釈
  ソース・デバッグ用の情報を出力しません。

[詳細説明]

- ソース・デバッグ用の情報を出力ファイル中に出力します。
- 本オプションを指定することにより，ソース・デバッグが可能となります。
- 最適化オプションと同時に指定した場合は，デバッグのしやすさに影響があります。

[使用例]

- ソース・デバッグ用の情報を出力ファイル中に出力します。

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

デバイス指定

デバイス指定オプションには，次のものがあります。

- -Xcommon
- -Xcpu

-Xcommon

デバイス共通のオブジェクト・ファイルを生成することを指定します。

[指定形式]

```
-Xcommon=series
```

- 省略時解釈
  エラーとなります。
  ただし，-V/-h/-P オプションを指定している場合は，エラーとなりません。
  CC-RH V1.02.00 以降は以下の通り変更します。
  命令セット・アーキテクチャを V850E3v5 とします（-Xcommon=v850e3v5 と同じ）。

[詳細説明]

- デバイス共通のオブジェクト・ファイルを生成することを指定します。
- 本オプションを指定した場合，ターゲットの命令セット・アーキテクチャの命令のみを使用し，また，命令セット・アーキテクチャに対応した共通のマジック・ナンバ *series* をオブジェクト・ファイルに埋め込みます。
- *series* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

v850e3v5	命令セット・アーキテクチャが V850E3V5 である品種をターゲット・デバイスとして指定したものとのリンクが可能です。
rh850	“v850e3v5” を指定した場合と同じです。

- *series* を省略した場合は，エラーとなります。

[使用例]

- 生成するオブジェクト・ファイルに，命令セット・アーキテクチャが V850E3V5 である品種に共通のマジック・ナンバを埋め込みます。

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

-Xcpu

指定したコア向けのオブジェクトを生成することを指定します。

[指定形式]

```
-Xcpu=core
```

- 省略時解釈
  -Xcommon オプションを指定した場合，G3M 向けオブジェクトを生成します。

[詳細説明]

- コア *core* 向けのオブジェクトを生成することを指定します。
- *core* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

g^{3m}	G3M 向けオブジェクトを生成します。
g^{3k}	G3K 向けオブジェクトを生成します。
g^{3mh}	G3MH 向けのオブジェクトを生成します。

- *core* を省略した場合は，エラーとなります。

[使用例]

- G3M 向けオブジェクトを生成することを指定します。

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

処理中断指定

処理中断指定オプションには，次のものがあります。

- -P
- -S
- -c

-P

入力ファイルに対してプリプロセス処理のみ実行します。

[指定形式]

```
-P
```

- 省略時解釈
  プリプロセス処理以降も処理を継続します。
  ファイルは出力しません。

[詳細説明]

- 入力ファイルに対してプリプロセス処理のみ実行して，結果をファイルに出力します。
- 出力ファイル名は，入力ファイル名の拡張子を “.i” に置き換えたものになります。
- -o オプションと同時に指定することにより，出力ファイル名を指定することができます。
- -Xpreprocess オプションを指定することにより，出力ファイルの内容を制御することができます。

[使用例]

- 入力ファイルに対してプリプロセス処理のみ実行して，結果をファイル main.i に出力します。

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

-S

アセンブル以降の処理を実行しません。

[指定形式]

```
-S
```

- 省略時解釈
  アセンブル以降も処理を継続します。

[詳細説明]

- アセンブル以降の処理を実行しません。
- ソース・ファイル名の拡張子を “.asm” で置き換えた名前でアセンブリ・ソース・ファイルを出力します。
- -o オプションと同時に指定することにより，出力ファイル名を指定することができます。

[使用例]

- アセンブル以降の処理を実行せず，アセンブリ・ソース・ファイル main.asm を出力します。

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

-c

リンク以降の処理を実行しません。

[指定形式]

```
-c
```

- 省略時解釈
  リンク以降も処理を継続します。

[詳細説明]

- リンク以降の処理を実行しません。
- ソース・ファイル名の拡張子を “.obj” で置き換えた名前でオブジェクト・ファイルを出力します。
- -o オプションと同時に指定することにより，出力ファイル名を指定することができます。

[使用例]

- リンク以降の処理を実行せず，オブジェクト・ファイル main.obj を出力します。

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

プリプロセッサ制御

プリプロセッサ制御オプションには，次のものがあります。

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

-D

プリプロセッサ・マクロ，およびアセンブラ・シンボルを定義します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- プリプロセッサ・マクロ，およびアセンブラのユーザ定義シンボルとして *name* を定義します。
- ソース・プログラムの前に，#define *name def*，および .SET *name def*（アセンブリ・ソース・プログラムの場合のみ）を記述するのと同等です。
- *name* が，アセンブラ・シンボルには使用可能であるがプリプロセッサ・マクロには使用できない文字（@，.，~）を含む場合，警告を出力してアセンブラ・シンボルとしてのみ定義します。
- 本オプションで C 言語の既定義マクロ __LINE__，__FILE__，__DATE__，__TIME__，__CCRH__ を再定義することはできません（-D__CCRH__[=1]，および -D__CCRH[=1] を除く）。
  入力ファイルが C ソース・ファイルの場合にこれらを再定義すると，エラーとなります。
- *name* を省略した場合は，エラーとなります。
- =*def* を省略した場合，*def* は 1 とみなします。
- 本オプションは，複数指定が可能です。
- 同じプリプロセッサ・マクロ，およびアセンブラ・シンボルに対して，本オプションと -U オプションを同時に指定した場合は，あとから指定したものが有効となります。

[使用例]

- プリプロセッサ・マクロとして sample=256 を定義します。

```
>ccrh -Dsample=256 -Xcommon=v850e3v5 main.c
```

-U

-D オプションによるプリプロセッサ・マクロ，およびアセンブラ・シンボルの定義を解除します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- -D オプションによるプリプロセッサ・マクロ，およびアセンブラのユーザ定義シンボル *name* の定義を解除します。
- ソース・プログラムの前に，#undef *name* を記述するのと同等です。
- *name* を省略した場合は，エラーとなります。
- 本オプションでは，#define *name def*，および .SET *name def*（アセンブリ・ソース・プログラムの場合のみ）の記述による定義は解除できません。
- 本オプションにより，C 言語の既定義マクロを解除することもできますが，__LINE__，__FILE__，__DATE__，__TIME__，__CCRH__，__CCRH を解除することはできません。
  入力ファイルが C ソース・ファイルの場合に *name* にこれらを指定すると，エラーとなります。
- 本オプションは，複数指定が可能です。
- 同じプリプロセッサ・マクロ，およびアセンブラ・シンボルに対して，本オプションと -D オプションを同時に指定した場合は，あとから指定したものが有効となります。

[使用例]

- -D オプションによるプリプロセッサ・マクロ test の定義を解除します。

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

-I

インクルード・ファイルを検索するフォルダを指定します。

[指定形式]

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

- 省略時解釈
  インクルード・ファイルを 標準インクルード・ファイル・フォルダからのみ検索します。

[詳細説明]

- プリプロセッサ指令 #include，およびアセンブラ制御命令 $INCLUDE/$BINCLUDE で読み込むインクルード・ファイルを検索するフォルダを *path* に指定します。
  インクルード・ファイルの検索は，以下の順番で行います。

(1) #include の場合

(a) ソース・ファイルのあるフォルダ（ファイルを ” ” で指定した場合）

(b) -I オプションで指定したフォルダ

(c) 標準インクルード・ファイル・フォルダ注

注 *本製品のインストール・フォルダ*¥CS+¥ccrh¥V*x.xx*¥inc

(2) $INCLUDE/$BINCLUDE の場合

(a) -I オプションで指定したフォルダ

(b) ソース・ファイルのあるフォルダ

(c) 元の C ソース・ファイルのあるフォルダ

(d) カレント・フォルダ

- *path* が存在しない場合は，警告を出力します。
- *path* を省略した場合は，エラーとなります。

[使用例]

- インクルード・ファイルをカレント・フォルダ，フォルダ D:¥include，標準フォルダの順で検索します。

```
>ccrh -ID:¥include -Xcommon=v850e3v5 main.c
```

-Xpreinclude

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。

[指定形式]

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

- 省略時解釈
  コンパイル単位の先頭にインクルードするファイルはないものとみなします。

[詳細説明]

- コンパイル単位の先頭にインクルードするファイルを *file* に指定します。

[使用例]

- コンパイル単位の先頭にファイル sample.h をインクルードします。

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

-Xpreprocess

プリプロセス結果の出力を制御します。

[指定形式]

```
-Xpreprocess=string[,string]
```

- 省略時解釈
  プリプロセス結果のファイルに，C ソースのコメント，および行番号情報を出力しません。

[詳細説明]

- プリプロセス結果のファイルに，C ソースのコメント，および行番号情報を出力します。
- 本オプションは，-P オプション指定時のみ有効です。
  -P オプションを指定しない場合は，本オプションを無視します。
- *string* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

comment	C ソースのコメントを出力します。
line	行番号情報を出力します。

< 行番号情報のフォーマット >

```
#line 行番号 " ファイル名 "
```

- *行番号*は 10 進数で，最大値は unsigned int の最大数となります。
- *ファイル名*はフルパスで，パス中の ¥ は ¥¥ に，" は ¥" に変換します。
  印字可能文字（スペースを含む）でない場合は，¥*3 桁の 8 進数* ("¥¥%03o") で出力します。
  改行文字は ¥¥n に変換します。
- 入力ソース・ファイル中に前処理指令 # *数値* " *文字列* "，または #line *数値* " *文字列* " を記述している場合は，*数値*を *行番号*，*文字列*を *ファイル名*として適用します。

- *string* を省略した場合は，エラーとなります。
- OS 標準の文字コードで出力します。

[使用例]

- プリプロセス結果のファイルに，C ソースのコメント，および行番号情報を出力します。

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

以下は上記の例と同等です。

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

C 言語制御

C 言語制御オプションには，次のものがあります。

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

-Xansi

C ソース・プログラムを ANSI 規格に厳密にあわせて処理します。

[指定形式]

```
-Xansi
```

- 省略時解釈
  従来の C 言語の仕様との両立性を持たせ，警告を出力して処理を続行します。

[詳細説明]

- C ソース・プログラムを ANSI 規格[注]に厳密にあわせて処理し，規格に反する記述に対してエラーや警告を出力します。
- 本オプション指定時は，マクロ名 __STDC__ を，値が 1 のマクロとして定義します。
- 言語仕様に厳密なコンパイル時の処理は，以下のようになります。
  - ビット・フィールド
    ビット・フィールドに int 型以外の型を指定した場合は，エラーとなります。
    本オプションを指定しない場合は，int 型以外の型の指定を許可します（警告は出力しません）。
  - # 行番号
    エラーとなります。
    本オプションを指定しない場合は，“#line 行番号 ” と同様に扱います。
  - #pragma inline された関数の引数
    指定した関数の呼び出しと定義の間で，戻り値の型や引数の型が異なるが型変換が可能である場合は，エラーとなります。
    本オプションを指定しない場合は，戻り値の型は呼び出し側の型に，引数は関数定義での型に変換して，インライン展開を行います。

注 ISO/IE C9899:1990（C90）で規程されている規格のことです。
ccrh は，ISO/IE C9899:1999（C99）で追加された仕様の一部も受容しますが，本オプションを指定した場合は，規格に反する記述はエラーとなります。

[使用例]

- C ソース・プログラムを ANSI 規格に厳密にあわせて処理し，規格に反する記述に対してエラーや警告を出力します。

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

-Xenum_type

列挙型に対して，どの整数型として扱うかを指定します。

[指定形式]

```
-Xenum_type=string
```

- 省略時解釈
  列挙型に対して，signed int として扱います。

[詳細説明]

- 列挙型に対して，どの整数型として扱うかを指定します。
- *string* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

auto	各列挙型について，その型のすべての列挙子の値を表現可能な最小の整数型として扱います。

- *string* を省略した場合は，エラーとなります。

[使用例]

- 各列挙型について，その型のすべての列挙子の値を表現可能な最小の整数型として扱います。

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

-Xvolatile

外部変数の volatile 化を指定します。

[指定形式]

```
-Xvolatile
```

- 省略時解釈
  volatile 修飾のある変数のみを volatile 宣言したものとして扱います。

[詳細説明]

- すべての外部変数を volatile 宣言したものとして扱います。
  外部変数のアクセス回数，アクセス順序は C ソース・ファイルで記述したとおりになります。

[使用例]

- すべての外部変数を volatile 宣言したものとして扱います。

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

-Xcheck

C ソース・ファイルの互換性をチェックします。

[指定形式]

```
-Xcheck=comp
```

- 省略時解釈
  C ソース・ファイルの互換性のチェックを行いません。

[詳細説明]

- *comp* に指定したコンパイラ用にコーディングした C ソース・ファイルを本コンパイラでコンパイルする際，互換性に影響するオプション指定，およびソース記述のチェックを行い，影響のある部分について警告，またはエラーを出力します。
- *comp* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

shc	SHC 用にコーディングした C ソース・ファイルをチェックします。

- *comp* を省略した場合は，エラーとなります。
- 主なチェック項目を以下に示します。
  - オプション：-Xbit_order=*pos*
    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。
    メッセージに出力されたオプションの選択を確認してください。
  - 拡張機能：#pragma section，#pragma entry，#pragma stacksize，#pragma address，#pragma global_register
    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。
    メッセージに出力された拡張仕様の記述を確認してください。
  - volatile 修飾した変数
    読み出しや書き込みのサイズがコンパイラ間で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いサイズでアクセスすることがありますが，SH コンパイラでは宣言型のサイズどおりにアクセスします。
  - 2 項演算の汎整数拡張
    unsigned int 型と long 型をオペランドとする 2 項演算（加減乗除や比較など）の結果が SH コンパイラと異なる場合があります。
    SH コンパイラでは -strict_ansi オプションを指定しない場合 signed long 型で演算します。
    本コンパイラではオペランドを unsigned int 型に変換してから演算します。
  - signed long 型を超える整数定数の型
    SH コンパイラでは unsigned long 型で表現可能な範囲の値を signed long long 型とします。
    本コンパイラでは unsigned long 型で表現可能な範囲の値は unsigned long 型とします。
  - ビット・フィールドの割り付け
    SH コンパイラではビット・フィールドの型が直前のビット・フィールドの型と異なる場合はビットを連続して配置しません。
    本コンパイラでは -Xpack オプションの指定に応じて連続して配置する場合があります。
- 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，メッセージを出力しません。
  割り付けを意識した宣言を行っている場合は，「CS+ 統合開発環境 ユーザーズマニュアル RH850 コーディング編」を参照してください。

［使用例］

- SH コンパイラ用にコーディングした C ソース・ファイルの互換性をチェックします。

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

-Xmisra2004

MISRA-C:2004 ルールによるソース・チェックを行います。

[指定形式]

```
-Xmisra2004=item[=value]
```

- 省略時解釈
  MISRA-C:2004 ルールによるソース・チェックを行いません。

[詳細説明]

- MISRA-C:2004 ルールによるソース・チェックを行います。
  指定したチェック項目 *item* に該当した場合，メッセージを出力します。
- *item* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

チェック項目 （*item*）	パラメータ （*value*）	説明
all	なし	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。
apply	*num*[,*num*]...	サポートしているルールのうち，*num* で指定した番号のルールをチェック対象とします。
ignore	*num*[,*num*]...	サポートしているルールのうち，*num* で指定した番号以外のルールをチェック対象とします。
required	なし	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。
required_add	*num*[,*num*]...	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 *num* で指定した番号のルールをチェック対象とします。
required_remove	*num*[,*num*]...	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 *num* で指定した番号を除いたルールをチェック対象とします。
file		サポートしているルールのうち，指定したファイル *file* に記載した番号のルールをチェック対象とします。 ファイル内では，1 行につき 1 ルール番号を指定します。

- *num* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。
  2.2 2.3
  4.1 4.2
  5.2 5.3 5.4 5.5 5.6
  6.1 6.2 6.3 6.4 6.5
  7.1
  8.1 8.2 8.3 8.5 8.6 8.7 8.11 8.12
  9.1 9.2 9.3
  10.1 10.2 10.3 10.4 10.5 10.6
  11.1 11.2 11.3 11.4 11.5
  12.1 12.3 12.4 12.5 12.6 12.7 12.8 12.9 12.10 12.11 12.12 12.13
  13.1 13.2 13.3 13.4
  14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9 14.10
  15.1 15.3 15.4 15.5
  16.1 16.3 16.5 16.6 16.9

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

- *item* を省略した場合は，エラーとなります。

［使用例］

- MISRA-C:2004 ルール番号 5.2，5.3，5.4 のルールをチェック対象としたソース・チェックを行います。

```
>ccrh -Xmisra2004=apply=5.2,5.3,5.4 -Xcommon=v850e3v5 main.c
```

-Xignore_files_misra

MISRA-C:2004 ルールによるソース・チェックの対象外のファイルを指定します。

[指定形式]

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

- 省略時解釈
  すべての C ソース・ファイルをチェック対象とします。

[詳細説明]

- *file* で指定したファイルを MISRA-C:2004 ルールによるソース・チェックの対象外とします。
- 本オプションは，-Xmisra2004 オプション指定時のみ有効です。
  -Xmisra2004 オプションを指定しない場合は，本オプションを無視します（警告は出力しません）。

[使用例]

- sample.c を MISRA-C:2004 ルールによるソース・チェックの対象外とします。

```
>ccrh -Xmisra2004=all -Xignore_files_misra=sample.c -Xcommon=v850e3v5 main.c
```

-Xcheck_language_extention

言語拡張により部分抑止される MISRA-C:2004 ルールのソース・チェックを有効にします。

[指定形式]

```
-Xcheck_language_extension
```

- 省略時解釈
  言語拡張により部分抑止される MISRA-C:2004 ルールのソース・チェックを無効にします。

[詳細説明]

- C 言語規格から独自に拡張した言語仕様のために MISRA-C:2004 ルールのソース・チェックが抑止される以下の場合について，ソース・チェックを有効にします。
  - プロトタイプ宣言がなく（ルール 8.1），該当関数に #pragma interrupt 指定がある場合
- 本オプションは，-Xmisra2004 オプション指定時のみ有効です。
  -Xmisra2004 オプションを指定しない場合は，本オプションを無視します（警告は出力しません）。

[使用例]

- 言語拡張により部分抑止される MISRA-C:2004 ルールのソース・チェックを有効にします。

```
>ccrh -Xmisra2004=all -Xcheck_language_extension -Xcommon=v850e3v5 main.c
```

日本語／中国語文字列制御

日本語／中国語文字列制御オプションには，次のものがあります。

- -Xcharacter_set

-Xcharacter_set

日本語／中国語の文字コードを指定します。

[指定形式]

```
-Xcharacter_set=code
```

- 省略時解釈
  日本語の文字コードを SJIS として扱います。

[詳細説明]

- ソース・ファイル中の日本語／中国語のコメント，文字列に対して，使用する文字コードを指定します。
- *code* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。
  なお，ソース・ファイル中で使用している文字コードと異なるものを指定した場合，動作は保証されません。

none	日本語／中国語の文字コードを解釈しません
euc_jp	EUC（日本語）
sjis	SJIS
utf8	UTF-8
big5	繁体字中国語
gb2312	簡体字中国語

- *code* を省略した場合は，エラーとなります。

[使用例]

- ソース・ファイル中の日本語のコメント，文字列に対して，使用する文字コードに EUC を指定します。

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

最適化指定

最適化指定オプションには，次のものがあります。

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

-O

最適化のレベル，または各最適化項目の詳細を指定します。

[指定形式]

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

- 省略時解釈
  デバッグに影響しない最適化のみを行います（-Odefault オプションの指定と同じです）。

[詳細説明]

- 最適化のレベル，または各最適化項目の詳細を指定します。
- *level* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

nothing	デバッグ優先の最適化 デバッグのしやすさを重視し，デフォルトで実行する最適化を含むすべての最適化を抑止します。
default	デフォルト デバッグに影響しない範囲の最適化（式の最適化，およびレジスタ割り付けなど）を行います。
size	オブジェクト・サイズ優先の最適化 ROM/RAM 容量の削減を重視して，一般的なプログラムに対して有効な最大限の最適化を行います。
speed	実行速度優先の最適化 実行速度の短縮を重視して，一般的なプログラムに対して有効な最大限の最適化を行います。

- *level* を省略した場合は，size を指定したものとみなします。
- *item*，および *value* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

最適化項目 （*item*）	パラメータ （*value*）	説明
unroll	0 ～ 4294967295 （整数値）	ループ展開 0，または 1 を指定した場合は，展開を抑止します。 2 以上を指定した場合は，実行回数が *N* 回（*N* は定数）のループと，*value* 回展開されたコードを含むループに変換します。 ただし，展開後のコード・サイズが大きいか，またはループの実行回数が少ない場合は，展開数が少なくなったり，展開しない場合があります。 また，内側にループを含むような複雑な構造のループは，展開しない場合があります。 *value* を省略した場合は，4 を指定したものとみなします。 本項目は，-Ospeed オプションを指定した場合は，-Ounroll=4 オプションを指定したものとみなします。

最適化項目 （*item*）	パラメータ （*value*）	説明
inline	0 ～ 3 （整数値）	関数のインライン展開 *value* は，展開のレベルを表します。 0：#pragma inline 指定した関数を含めて，すべてのインライン展開を抑止します。 1：#pragma inline 指定した関数のみ展開します。 2：自動的に展開対象の関数を判別して展開します。 3：コード・サイズがなるべく増加しない範囲で，自動的に展開対象の関数を判別して展開します。 ただし，1 ～ 3 を指定した場合でも，関数の内容やコンパイル状況により，#pragma inline 指定した関数が展開されない場合があります。 *value* を省略した場合は，2 を指定したものとみなします。 本項目は，-Osize，または -Ospeed オプションを指定した場合に有効です（-Osize オプションを指定した場合は -Oinline=3，-Ospeed オプションを指定した場合は -Oinline=2 オプションを指定したものとみなします）。 -Osize，-Ospeed，-Oinline オプションのいずれも指定していない場合は，-Oinline=1 オプション指定したものとみなします。 -Onothing オプションを指定した場合は，-Oinline=0 オプションを指定したものとみなします。
inline_size	0 ～ 65535 （整数値）	インライン展開サイズ コード・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します。 *value* を省略した場合は，100 を指定したものとみなします。 本項目は，-Oinline=2 オプションを指定した場合に有効です（-Ospeed オプションによる指定も含みます）。
delete_static_func	on，または off	未使用 static 関数の削除 *value* を省略した場合は，on を指定したものとみなします。 本項目は，-Onothing オプション指定時以外に有効です。
pipeline	on，または off	パイプライン最適化 *value* を省略した場合は，on を指定したものとみなします。 本項目は，-Ospeed オプション指定時は，常に有効です。
tail_call	on，または off	関数末尾の関数呼び出しの jr 化 on を指定した場合，関数の末尾が関数呼び出しであり，かつ一定の条件を満たす場合に，その呼び出しに対して jarl 命令ではなく jr 命令を生成して lp の退避／復帰コードを削除し，コード・サイズを削減します。 ただし，一部のデバッグ機能を使用することができなくなります。 *value* を省略した場合は，on を指定したものとみなします。 本項目は，-Ospeed，または -Osize オプションを指定した場合に有効です。
map	*ファイル名*	外部変数アクセス最適化 最適化リンカ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，また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。 また，スタートアップ・ルーチンでシンボル “__gp_data” を定義して gp に値を設定するコードを記述した場合，外部シンボル割り付け情報を元に，可能であれば変数のアクセスを gp 相対で行うコードを生成します。 *ファイル名*には，最適化リンカが生成した外部シンボル割り付け情報ファイルを指定します。 *ファイル名*を省略した場合は，一度リンク処理まで実行し，外部シンボル割り付け情報ファイルを作成したのち，再度コンパイルからリンクまでの処理を行います。

最適化項目 （*item*）	パラメータ （*value*）	説明
smap	なし	コンパイル単位内で定義された外部変数に対する外部変数アクセス最適化 コンパイル対象ファイル内で定義された外部変数，また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。

- *item* を省略した場合，本オプションは -Osize を指定したものとみなします。
- 同じ *item* について，本オプションを複数指定した場合は，あとから指定したものが有効となります。
- -O*item* を指定しなかった場合の最適化項目は，-O*level* の指定によって以下のように解釈します。

最適化項目 （*item*）	最適化レベル（*level*）			
	nothing	default	size	speed
unroll	1	1	1	4
inline	0	1	3	2
inline_size	—	—	—	100
delete_static_func	off	on	on	on
pipeline	off	off	off	on
tail_call	off	off	on	on
map	—	—	—	—
smap	—	—	—	—

［使用例］

- オブジェクト・サイズ優先の最適化を行います。

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

-Xintermodule

大域最適化を行います。

[指定形式]

```
-Xintermodule
```

- 省略時解釈
  大域最適化を行いません。

[詳細説明]

- 大域最適化を行います。
- 主な最適化内容を以下に示します。
  - 手続き間別名解析を利用した最適化
    出力コード例を以下に示します。

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

[ 出力アセンブラ・ソース ]
_func1.1:
    .stack  _func1.1 = 0
    mov     #_x, r2
    st.w    r0, 0x00000000[r2]
    mov     0x00000001, r5
    st.w    r5, 0x00000004[r2]
    mov     0x00000000, r10     ; a と b の指す先が違うため 0 を直接代入
    jmp     [r31]
_func2:
    .stack  _func2 = 0
    mov     #_x, r6
    addi    0x00000004, r6, r7
    br9     _func1.1
```

- パラメータ，戻り値の定数伝播
  出力コード例を以下に示します。

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

[ 出力アセンブラ・ソース ]
_func.1:
    .stack  _func.1 = 0
    mov     0x00000004, r10     ; 4(=3-4+5) を直接代入
    jmp     [r31]
_func2:
    .stack  _func2 = 0
    mov     0x00000005, r8
    mov     0x00000004, r7
    mov     0x00000003, r6
    br9     _func.1
```

［使用例］

- ソース・ファイル main.c，sub.c の大域最適化を行います。

```
>ccrh -Xintermodule -Xwhole_program -Xcommon=v850e3v5 main.c sub.c
```

-Xinline_strcpy

標準ライブラリ関数 strcpy，strcmp，memcpy，memset の呼び出しをインライン展開します。

[指定形式]

```
-Xinline_strcpy
```

- 省略時解釈
  標準ライブラリ関数 strcpy，strcmp，memcpy，memset のインライン展開を行いません。

[詳細説明]

- 標準ライブラリ関数 strcpy，strcmp，memcpy，memset の呼び出しをインライン展開します。
- 本オプションは，-Xpack オプションと同時に指定することはできません。
- strcpy については，第二引数が文字列の場合にのみ，インライン展開を行います。
- 本オプションを指定した場合，配列，および文字列は自動的に 4 バイト境界に配置されます。
- 生成されるプログラムの実行速度は高速になりますが，コード・サイズは増大します。

[使用例]

- 標準ライブラリ関数 strcpy，strcmp，memcpy，memset の呼び出しをインライン展開します。

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

-Xmerge_string

文字列定数をマージします。

[指定形式]

```
-Xmerge_string
```

- 省略時解釈
  ソース・ファイル内で同じ文字列定数が複数存在する場合，それぞれを別々の領域に割り付けます。

[詳細説明]

- ソース・ファイル内で同じ文字列定数が複数存在する場合，これらをまとめて 1 つの領域に割り付けます。
- #pragma section sconst（const）の指定に依らず，同じ文字列定数を同一領域に割り付けます。
  ただし，異なるセクションを指定した場合，文字列定数の割り付け先のセクションはソース中での出現順序に依存します。

[使用例]

- ソース・ファイル内で同じ文字列定数が複数存在する場合，これらをまとめて 1 つの領域に割り付けます。

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

-Xalias

ポインタ指示先の型を考慮した最適化の指定を行います。

[指定形式]

```
-Xalias=value
```

- 省略時解釈
  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。

[詳細説明]

- 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うかどうかを指定します。
- *value* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

ansi	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。
noansi	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。

- *value* を省略した場合は，エラーとなります。

[使用例]

- 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。

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

-Xmerge_files

複数の C ソース・ファイルをマージしてコンパイルを行います。

[指定形式]

```
-Xmerge_files
```

- 省略時解釈
  マージせずに入力ファイル単位でコンパイルを行います。

[詳細説明]

- 複数の C ソース・ファイルをマージしてコンパイルを行い，1 ファイルで出力します。
- 出力ファイル名は，-o オプションを指定した場合は指定したファイル名となり，-o オプションを指定しなかった場合は最初に指定した C ソース・ファイル名に対して -o オプションの省略時解釈に従ったファイル名となります。
- 本オプションは，入力 C ソース・ファイルが 1 ファイルの場合，および -P オプションと同時に指定した場合は無効となります。
- 本オプションは，-S，または -c オプションと同時に指定した場合は，2 番目以降に指定した C ソース・ファイル名について -o オプションの省略時解釈に従ったファイル名の空ファイルを出力します。
- -Oinline オプションと同時に指定した場合は，ファイル間インライン展開を行います。
- 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 のいずれかを同時に指定した場合，動作は保証されません。

[使用例]

- main.c，sub.c をマージしてコンパイルを行い，1 ファイルで出力します。

```
>ccrh -Xmerge_files -Xwhole_program -Xcommon=v850e3v5 main.c sub.c
```

-Xwhole_program

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て最適化を行います。

[指定形式]

```
-Xwhole_programTFE
```

- 省略時解釈
  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ません。

[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て最適化を行います。
- 以下の条件を満たすことを前提にコンパイルを行い，条件を満たさなかった場合の動作は保証されません。
  - コンパイル対象ファイル内で定義した extern 変数の値，およびアドレスが，コンパイル対象ファイル以外で変更，および参照されない。
  - 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で定義した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がない。
- 本オプションを指定した場合，-Xintermodule オプションを指定したものとみなしてコンパイルを行います。
  また，入力 C ソース・ファイルが複数の場合，-Xmerge_files オプションを指定したものとみなしてコンパイルを行います。

[使用例]

- 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て最適化を行います。

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

生成コード制御

生成コード制御オプションには，次のものがあります。

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

-Xpack

構造体パッキングを行います。

[指定形式]

```
-Xpack=num
```

- 省略時解釈
  構造体パッキングを行いません。

[詳細説明]

- 構造体パッキングを行います。
- 本オプションを指定した場合，構造体のメンバをその型でアライメントせず，指定した *num* バイトのアライメントに詰めてコードを生成します。
- *num* には，1，2，4 のいずれかを指定することができます。
  これ以外のものを指定した場合は，エラーとなります。
- *num* を省略した場合は，エラーとなります。
- 本オプションは，-Xinline_strcpy オプションと同時に指定することはできません。
- C ソース中に #pragma 指令で構造体パッキングを指定している場合に本オプションを指定すると，最初の #pragma 指令が出現するまでは，オプションの指定値をすべての構造体に適用します。
  それ以降は，#pragma 指令の値を適用します。
  ただし，#pragma 指令の出現後でも，指定がデフォルトになった部分（#pragma 指令でパッキング値を指定しない場合）は，オプションの指定値を適用します。

[使用例]

- 構造体のメンバを 1 バイトのアライメントに詰めてコードを生成します。

```
>ccrh -Xpack=1 -Xcommon=v850e3v5 main.c
```

-Xbit_order

ビット・フィールドのメンバの並び順を指定します。

[指定形式]

```
-Xbit_order=pos
```

- 省略時解釈
  ビット・フィールドのメンバを下位ビットから割り付けます。

[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。
- *pos* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

left	上位ビットからメンバを割り付けます。
right	下位ビットからメンバを割り付けます。

- *pos* を省略した場合は，エラーとなります。
- C ソース中に #pragma 指令でビット・フィールドのメンバの並び順を指定している場合，最初の #pragma bit_order 指令が出現するまでは，オプション指定値がすべてのビット・フィールドのメンバに適用されます。
  それ以降は，#pragma 指令の値が適用されます。

[使用例]

- ビット・フィールドのメンバを上位ビットから割り付けます。

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

-Xpass_source

出力するアセンブリ・ソース・ファイル中に C ソース・プログラムをコメントとして出力します。

[指定形式]

```
-Xpass_source
```

- 省略時解釈
  出力するアセンブリ・ソース・ファイル中に C ソース・プログラムをコメントとして出力しません。

[詳細説明]

- 出力するアセンブリ・ソース・ファイル中に C ソース・プログラムをコメントとして出力します。
- 出力するコメントは，あくまで参考であり，厳密にはコードと対応していない場合もあります。
  また，実行文以外の行にはコメントとして出力されないものもあります（型宣言やラベルなど）。
  たとえば，グローバル変数とローカル変数，関数宣言などのコメントの出力位置がずれることがあります。
  また，最適化オプションを指定した場合などにより，コードが削除され，コメントのみが残ることもあります。

[使用例]

- 出力するアセンブリ・ソース・ファイル中に C ソース・プログラムをコメントとして出力します。

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

-Xswitch

switch 文のコード出力形式を指定します。

[指定形式]

```
-Xswitch=type
```

- 省略時解釈
  コンパイラが switch 文ごとに最適な出力形式を 自動的に選択します。

[詳細説明]

- switch 文のコード出力形式を指定します。
- *type* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

ifelse	case 文の並びに沿った if-else 文と同じ形式で出力します。 頻度が多い順に case 文を書いているときやラベル数が少ないときに，本項目を指定します。
binary	バイナリ・サーチ形式で出力します。 バイナリ・サーチ・アルゴリズムに用いて合致する case 文を探します。 ラベル数が多いときに本項目を選択すると，どの case 文も同じくらいの速さで見つけることができます。
table	テーブル・ジャンプ形式で出力します。 case 文の値を基にインデックス化したテーブルを参照し，switch 文の値により case ラベルを選択して処理を行います。 どの case 文にも同じくらい速く分岐します。 ただし，case 値が連続していないときは無駄な領域ができます。 また，case ラベルの最大値と最小値の差が 8192 を超える場合は，本オプションを無視して，switch 文ごとに最適な出力形式を 自動的に選択します。

- *type* を省略した場合は，エラーとなります。

[使用例]

- switch 文のコードに対して，バイナリ・サーチ形式で出力します。

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

-Xreg_mode

レジスタ・モードを指定します。

[指定形式]

```
-Xreg_mode=mode
```

- 省略時解釈
  32 レジスタ・モードのオブジェクト・ファイルを生成します。

[詳細説明]

- 指定したレジスタ・モードのオブジェクト・ファイルを生成します。
- ccrh が使用するレジスタを 32 本（32 レジスタ・モード），22 本（22 レジスタ・モード，または common レジスタ・モード）のいずれかに制限し，オブジェクト・ファイル内にレジスタ・モードを示すマジック・ナンバを埋め込みます。
- common レジスタ・モードは，レジスタ・モードに依存しないオブジェクト・ファイルを生成するために使用します。
- *mode* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

レジスタ・モード（*mode*）	作業用レジスタ	レジスタ変数用レジスタ
common	r10 ～ r14	r25 ～ r29
22	r10 ～ r14	r25 ～ r29
32	r10 ～ r19	r20 ～ r29

- *mode* を省略した場合は，エラーとなります。
- C ソース・ファイルに対して，使用可能なレジスタのみを用いたコードを生成します。
- アセンブリ・ソース・ファイルに対して，使用可能なレジスタ以外のレジスタが使用されている場合は，エラーとなります。
- 本オプションは，すべてのソース・ファイルに対して同じレジスタ・モードを指定する必要があります。
  ソース・ファイルごとに指定を変えることはできません。
  レジスタ・モードが異なるオブジェクト・ファイルが混在している場合は，リンク時にエラーとなります。
- 本オプションを指定することにより，ソフトウエア・レジスタ・バンク機能のレジスタ・モードを切り替えることができます。

[使用例]

- 22 レジスタ・モードのオブジェクト・ファイルを生成します。

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

-Xreserve_r2

r2 レジスタを予約します。

[指定形式]

```
-Xreserve_r2
```

- 省略時解釈
  r2 レジスタの予約を行わずに，コンパイラで使用します。

[詳細説明]

- r2 レジスタを予約し，コンパイラでは r2 レジスタを使用しないコードを生成します。

[使用例]

- r2 レジスタを予約し，コンパイラでは r2 レジスタを使用しないコードを生成します。

```
>ccrh -Xreserve_r2 -Xcommon=v850e3v5 main.c
```

-Xep

ep レジスタの扱い方を指定します。

[指定形式]

```
-Xep=mode
```

- 省略時解釈
  ep レジスタを関数呼び出し前後で値を保証するレジスタとして扱います。

[詳細説明]

- ep レジスタの扱い方を指定します。
- *mode* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

fix	ep レジスタの値をプロジェクト全体で固定します。 プロジェクト内の #pragma section 指令において，以下の属性指定文字を使用した場合は，本パラメータを指定してください。 ep_auto，ep_disp4，ep_disp5，ep_disp7，ep_disp8，ep_disp16，ep_disp23
callee	ep レジスタを関数呼び出し前後で値を保証するレジスタとして扱います。 -Omap，または -Osmap オプションを指定した場合は，本パラメータを指定してください。

- *mode* を省略した場合は，エラーとなります。

[使用例]

- ep レジスタの値をプロジェクト全体で固定します。

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

-Xfloat

浮動小数点演算命令の生成を制御します。

[指定形式]

```
-Xfloat=type
```

- 省略時解釈
  -Xcpu=g3m オプションを指定した場合，および -Xcpu オプションを指定しなかった場合は，浮動小数点演算命令を生成します。
  -Xcpu=g3k オプションを指定した場合は，ランタイム関数の呼び出し命令を生成します。

[詳細説明]

- 浮動小数点演算命令の生成を制御します。
- *type* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

soft	浮動小数点演算に対して，ランタイム・ライブラリの呼び出し命令を生成します。 また，FPU 数学対応ライブラリはリンクしません。
fpu	浮動小数点演算に対して，FPU（浮動小数点ユニット）の浮動小数点演算命令を生成します。 また，FPU 数学対応ライブラリをリンクします。

- *type* を省略した場合は，エラーとなります。
- 本オプションは，-Xcpu=g3k オプションと同時に指定した場合は無効となり，常にランタイム関数の呼び出し命令を生成します。
- *type* に soft を指定した場合，-Xround=zero オプションは無効となり，常に -Xround=nearest が有効となります。

[使用例]

- 浮動小数点演算に対して，ランタイム・ライブラリの呼び出し命令を生成します。

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

-Xcall_jump

関数呼び出しの分岐命令の生成を制御します。

[指定形式]

```
-Xcall_jump=num
```

- 省略時解釈
  関数呼び出しの分岐に対して，jarl，および jr 命令を生成します。

[詳細説明]

- 関数呼び出しの分岐命令の生成を制御します。
- *num* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

22	関数呼び出しの分岐に対して，jarl，および jr 命令を生成します。
32	関数呼び出しの分岐に対して，jarl32，および jr32 命令を生成します。

- *num* を省略した場合は，エラーとなります。

[使用例]

- 関数呼び出しの分岐に対して，jarl32，および jr32 命令を生成します。

```
>ccrh -Xcall_jump=32 -Xcommon=v850e3v5 main.c
```

-Xfar_jump

far jump の出力を制御します。

[指定形式]

```
-Xfar_jump=file
```

- 省略時解釈
  -Xcall_jump オプションに従った命令を生成します。

[詳細説明]

- C ソース・ファイルについて，far jump 呼び出し関数一覧ファイル *file* 中で指定した関数（C 言語で記述した割り込み関数を含む）への分岐に対して，jarl32，および jr32 命令を生成します。
- *file* の推奨拡張子は，“.fjp” です。
- *file* が存在しない場合は，エラーとなります。
- *file* を省略した場合は，エラーとなります。
- -Xcall_jump=22 オプションを指定した際，関数本体が jarl，および jr 命令では分岐できない範囲（±2M バイト以上）にあり，リンク時にエラーとなる場合，本オプションを使用してコンパイルし直します。
- 本オプションを複数指定した場合は，最後に指定したものが有効となります。
- 出力コード例を以下に示します。
  - C ソース

```
far_func();     /* デフォルトでは jarl _far_func, lp を出力する */
```

  - 出力アセンブリ・ソース

```
jarl32  _far_func, lp
```

備考　far jump 呼び出し関数一覧ファイルの書式に関する注意事項を以下に示します。

- 1 行に 1 個の関数名を記述します。
  複数の関数名を記述した場合は，最初の名前のみを有効とします。
- 記述する関数名は，C 言語の関数名の先頭に “_” を付けた名前（アセンブリ・ソースでのラベル名）とします。
  ただし，関数名の代わりに，以下の形式で指定することもできます。

形式	意味
{all function}	すべての関数呼び出しを対象とします。
{all interrupt}	すべての割り込み関数呼び出しを対象とします。

- 関数だけでなく，ランタイム・ルーチンも指定可能です。
  その場合は，関数名としてランタイム・ルーチン名をそのまま記述してください。
- 関数名の前後に，空白やタブを挿入することが可能です。
- 使用可能な文字は，ASCII 文字のみです。
  なお，行頭の空白を除いたあと，次の空白，または行末までの非空白文字列を関数名とし，空白から行末までは無視します。

- コメントを挿入することはできません。
- 1 行に記述可能な文字数は，1023 文字までです（空白やタブも含みます）。

関数の指定例を以下に示します。

```
_func_led
_func_beep
_func_motor
     :
_func_switch
```

［使用例］

- func.fjp 中で指定した関数への分岐に対して，jmp 命令を使用したコードを生成します。

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

-Xdiv

除算に対して，div，および divu 命令を生成します。

[指定形式]

```
-Xdiv
```

- 省略時解釈
  除算に対して，divq，および divqu 命令を生成します。

[詳細説明]

- 除算に対して，divq，および divqu 命令を生成する代わりに，div，および divu 命令を生成します。
- divq，および divqu 命令は高速ですが，実行サイクル数がオペランドの値により変わります。
  そのため，リアルタイム性の保証などのために，常に実行サイクル数が一定であることが必要な場合に，本オプションを指定します。

[使用例]

- 除算に対して，div，および divu 命令を生成します。

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

-Xcheck_div_ov

除算時に OV フラグのチェックを行います。

[指定形式]

```
-Xcheck_div_ov=num
```

- 省略時解釈
  除算時に OV フラグのチェックを行わないコードを生成します。

[詳細説明]

- 除算命令の後に OV フラグのチェックを行い，OV フラグが 1 のときに FE レベル・ソフトウエア例外を発生させるコード（fetrap 命令）を生成します。
- *num* に指定可能な値は，1 ～ 15（fetrap 命令のオペランドで指定可能な値）です。
  これ以外のものを指定した場合は，エラーとなります。
- *num* を省略した場合は，エラーとなります。

[使用例]

- 除算時に OV フラグのチェックを行います。

```
>ccrh -Xcheck_div_ov=1 -Xcommon=v850e3v5 main.c
```

-Xuse_fmaf

積和演算命令を生成します。

[指定形式]

```
-Xuse_fmaf
```

- 省略時解釈
  積和演算命令を生成しません。

[詳細説明]

- 単精度浮動小数点積和演算に対して，積和演算命令（fmaf.s，fmsf.s，fnmaf.s，fnmsf.s）を生成します。
- 本オプションを指定すると，実行速度は上がりますが，演算精度が変わります。

[使用例]

- 単精度浮動小数点積和演算に対して，積和演算命令を生成します。

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

-Xunordered_cmpf

浮動小数点比較において，無効演算例外の検出を行います。

[指定形式]

```
-Xunordered_cmpf
```

- 省略時解釈
  浮動小数点比較において，無効演算例外の検出を行いません。

[詳細説明]

- 浮動小数点比較において，比較する値に非数が含まれている場合に無効演算例外を発生する比較条件を使用するコードを生成します。

[使用例]

- 浮動小数点比較において，無効演算例外の検出を行います。

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

-Xmulti_level

マルチコア用プログラムの生成を指定します。

[指定形式]

```
-Xmulti_level=level
```

- 省略時解釈
  シングルコア用プログラムを生成します。

[詳細説明]

- 指定したコア用のプログラムを生成します。
- *level* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

0	シングルコア用プログラムを生成します。 プログラム中の #pragma pmodule 指定は無視します。
1	マルチコア用プログラムを生成します。 プログラム中の #pragma pmodule 指定は有効となります。

[使用例]

- マルチコア用プログラムを生成します。

```
>ccrh -Xmulti_level=1 -Xcommon=v850e3v5 file1.c file2.c
```

-Xpatch

パッチを適用します。

[指定形式]

```
-Xpatch=string
```

- 省略時解釈
  パッチを適用しません。

[詳細説明]

- *string* に指定可能なものは，“dw_access” です。
  これ以外のものを指定した場合は，エラーとなります。
- -Xpatch=dw_access オプションを指定した場合，ld.dw/st.dw 命令の生成を抑止したコードを生成します。

[使用例]

- ld.dw/st.dw 命令の生成を抑止したコードを生成します。

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

-Xdbl_size

double 型および long double 型の精度を指定します。

[指定形式]

```
-Xdbl_size=num
```

- 省略時解釈
  double 型および long double 型を倍精度浮動小数点型 (8 バイト) として扱う (-Xdbl_size=8 と同じ)。

[詳細説明]

- double 型および long double 型の精度を指定します。
- num には以下のいずれかを指定します。これ以外のものを指定した場合はエラーとなります。
  - 4
    double 型および long double 型を単精度浮動小数点型 (4 バイト) として扱う。
  - 8
    double 型および long double 型を倍精度浮動小数点型 (8 バイト) として扱う。

-Xround

浮動小数点定数の丸めモードを指定します。

[指定形式]

```
-Xround=mode
```

- 省略時解釈
  浮動小数点定数を表現可能な最も近い値に丸めます（-Xround=nearest と同じ）。

[詳細説明]

- 浮動小数点定数の丸めモードを指定します。
- mode には以下のいずれかを指定します。これ以外のものを指定した場合はエラーとなります。
  - nearest
    浮動小数点定数を表現可能な最も近い値に丸めます。
  - zero
    浮動小数点定数を 0 の方に丸めます。
- -Xfloat=soft と同時に指定した場合，本オプションは無効になり，常に浮動小数点定数を表現可能な最も近い値に丸めます。

-Xalign4

関数の先頭アドレスのアライメントを指定します。

[指定形式]

```
-Xalign4
```

- 省略時解釈
  関数の先頭アドレスのアライメントを 2 にします。

[詳細説明]

- 関数の先頭アドレスのアライメントを 4 にします。
- 本オプションを使用しないでコンパイルして生成したオブジェクト・モジュール・ファイルや標準ライブラリとリンクした場合，リンク時に W0561322 の警告を出力しますが，動作は問題ありません。

-Xsection

データのデフォルトセクションを指定します。

[指定形式]

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

- 省略時解釈
  初期値なしデータは .bss，初期値ありデータは .data，定数データは .const をデフォルトとします。

[詳細説明]

- データのデフォルトセクション属性を指定します。
- *string* と *value* に指定可能な文字列と，各々の場合のデフォルトセクションは以下の通りです。*string* と *value* にこれ以外のものを指定した場合はエラーとなります。

string	value	デフォルトセクション		
		初期値なしデータ	初期値ありデータ	定数データ
data	r0_disp16	.zbss	.zdata	-
	r0_disp23	.zbss23	.zdata23	-
	ep_disp16	.ebss	.edata	-
	ep_disp23	.ebss23	.edata23	-
	gp_disp16	.sbss	.sdata	-
	gp_disp23	.sbss23	.sdata23	-
const	zconst	-	-	.zconst
	zconst23	-	-	.zconst23

- #pragma section で属性を変更した場合は，#pragma section で指定した属性が有効になります。
  value に ep_disp16 もしくは ep_disp23 を指定した場合，-Xep=fix が指定されたとみなしてコンパイルし，-Omap もしくは -Osmap と同時に使用するとエラーとなります。

情報ファイル出力制御

情報ファイル出力制御オプションには，次のものがあります。

- -Xcref

-Xcref

静的解析情報ファイルを出力します。

[指定形式]

```
-Xcref=path
```

- 省略時解釈
  静的解析情報ファイルを出力しません。

[詳細説明]

- コンパイル途中に生成される静的解析情報ファイルの保存先を *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，C ソース・ファイル名の拡張子を “.cref” で置き換えたファイル名で静的解析情報ファイルを保存します。
- *path* に存在するファイル名を指定した場合，および存在しないフォルダ名，またはファイル名を指定した場合は，出力する静的解析情報ファイルが 1 つの場合は，*path* というファイル名で保存します。
  出力する静的解析情報ファイルが複数の場合は，エラーとなります。
- =*path* を省略した場合は，エラーとなります。
- ソース・ファイルとして同じ名前のファイル（異なるフォルダにある場合を含む）を複数指定した場合は，警告を出力して，最後に指定したソース・ファイルに対する静的解析情報ファイルのみを保存します。

[使用例]

- 静的解析情報ファイルをファイル名 info.cref で出力します。

```
>ccrh -Xcref=info.cref -Xcommon=v850e3v5 main.c
```

エラー出力制御

エラー出力制御オプションには，次のものがあります。

- -Xerror_file

-Xerror_file

エラー・メッセージをファイルに出力します。

[指定形式]

```
-Xerror_file=file
```

- 省略時解釈
  エラー・メッセージを標準エラー出力のみに出力します。

[詳細説明]

- エラー・メッセージを標準エラー出力，およびファイル *file* に出力します。
- *file* がすでに存在する場合は，そのファイルを上書きします。
- *file* を省略した場合は，エラーとなります。

[使用例]

- エラー・メッセージを標準エラー出力，およびファイル err に出力します。

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

警告メッセージ出力制御

警告メッセージ出力制御オプションには，次のものがあります。

- -Xno_warning

-Xno_warning

指定した警告メッセージの出力を抑止します。

[指定形式]

```
-Xno_warning=num[,num]...
-Xno_warning=num1-num2
```

- 省略時解釈
  すべての警告メッセージを出力します。

[詳細説明]

- 指定した警告メッセージの出力を抑止します。
- *num*，*num1*，*num2* には，メッセージ番号を指定します。
  存在しないメッセージ番号を指定した場合は，無視します。
- *num*，または *num1*，および *num2* を省略した場合は，エラーとなります。
- *num1-num2* の形式で指定すると，その範囲に含まれるメッセージ番号を指定したものとみなします。
- 本オプションで指定するメッセージ番号は，W に続く 7 桁の数字のうち，下位 5 桁です。
  メッセージ番号については，「CS+ 統合開発環境 ユーザーズマニュアル メッセージ編」を参照してください。
- 本オプションで制御することができるのは，コンパイラ，アセンブラの警告メッセージです（最適化リンカの警告メッセージは対象外です）。

[使用例]

- 警告メッセージ W0511146，W0511147 の出力を抑止します。

```
>ccrh -Xno_warning=11146,11147 -Xcommon=v850e3v5 main.c
```

フェーズ個別オプション指定

フェーズ個別オプション指定オプションには，次のものがあります。

- -Xasm_option
- -Xlk_option

-Xasm_option

アセンブル・オプションを指定します。

[指定形式]

```
-Xasm_option=arg
```

- 省略時解釈
  指定したオプションはすべて ccrh のドライバが解釈します。

[詳細説明]

- *arg* をアセンブル・オプションとして，アセンブラに渡します。
- ccrh のドライバがアセンブリ・ソース・ファイルとして認識しない識別子を持つファイルをアセンブラに渡す際には，本オプションを使用します。
- *arg* が存在しないアセンブル・オプションである場合は，エラーとなります。
- *arg* を省略した場合は，エラーとなります。

[使用例]

- -Xasm_far_jump オプションをアセンブラに渡します。

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

上記の例は，以下と同じ意味となります。

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

-Xlk_option

リンク・オプションを指定します。

[指定形式]

```
-Xlk_option=arg
```

- 省略時解釈
  指定したオプションはすべて ccrh のドライバが解釈します。

[詳細説明]

- *arg* をリンク・オプションとして，最適化リンカに渡します。
- ccrh のドライバが最適化リンカへの入力として認識しない識別子を持つファイルを最適化リンカに渡す際には，本オプションを使用します。
- *arg* が存在しないリンク・オプションである場合は，エラーとなります。
- *arg* を省略した場合は，エラーとなります。

[使用例]

- -form=relocate オプションを最適化リンカに渡します。

```
>ccrh -Xlk_option=-form=relocate -Xcommon=v850e3v5 main.c
```

上記の例は，以下と同じ意味となります。

```
>ccrh -c -Xcommon=v850e3v5 main.c
>rlink -form=relocate main.obj
```

サブコマンド・ファイル指定

サブコマンド・ファイル指定オプションには，次のものがあります。

- @

@

サブコマンド・ファイルを指定します。

[指定形式]

```
@file
```

- 省略時解釈
  コマンド・ラインで指定したオプション，およびファイル名のみを認識します。

[詳細説明]

- *file* をサブコマンド・ファイルとして扱います。
- *file* が存在しない場合は，エラーとなります。
- *file* を省略した場合は，エラーとなります。
- サブコマンド・ファイルについての詳細は，「2.4.2 サブコマンド・ファイルの使用方法」を参照してください。

[使用例]

- command.txt をサブコマンド・ファイルとして扱います。

```
>ccrh @command.txt -Xcommon=v850e3v5
```

2.5.2 アセンブル・オプション

ここでは，アセンブル・フェーズのオプションについて説明します。

オプションに関する注意事項を以下に示します。

- オプションの大文字／小文字は区別します。
- パラメータとして数値を指定する場合は，10 進数，または “0x”（“0X”）で始まる 16 進数での指定が可能です。16 進数のアルファベットは，大文字／小文字を区別しません。
- パラメータとしてファイル名を指定する場合は，パス付き（絶対パス，または相対パス）での指定が可能です。パスなし，および相対パスで指定する場合は，カレント・フォルダを基準とします。
- パラメータ中に空白を含める場合（パス名など）は，そのパラメータ全体をダブルクォーテーション（"）で囲んでください。
- ccrh コマンドに対して，-Xprn_path，または -Xasm_far_jump オプションを指定する場合は，-Xasm_option オプションを使用する必要があります。

オプションの分類と説明を以下に示します。

表 2.3 アセンブル・オプション

分類	オプション	説明
バージョン／ヘルプ表示指定	-V	asrh のバージョン情報を表示します。
	-h	asrh のオプションの説明を表示します。
出力ファイル指定	-o	出力ファイル名を指定します。
	-Xobj_path	アセンブル途中に生成されるオブジェクト・ファイルの保存先を指定します。
	-Xprn_path	アセンブル・リスト・ファイルの保存先を指定します。
ソース・デバッグ制御	-g	ソース・デバッグ用の情報を出力します。
デバイス指定	-Xcommon	デバイス共通のオブジェクト・ファイルを生成することを指定します。
	-Xcpu	指定したコア向けのオブジェクトを生成することを指定します。
プリプロセッサ制御	-D	アセンブラ・シンボルを定義します。
	-U	-D オプションによるアセンブラ・シンボルの定義を解除します。
	-I	インクルード・ファイルを検索するフォルダを指定します。
日本語／中国語文字列制御	-Xcharacter_set	日本語／中国語の文字コードを指定します。
生成コード制御	-Xreg_mode	レジスタ・モードを指定します。
	-Xreserve_r2	r2 レジスタを予約します。
	-Xep	ep レジスタの扱い方を指定します。
アセンブラ制御指定	-Xasm_far_jump	アセンブリ・ソース・ファイルに対して，far jump の出力を制御します。
エラー出力制御	-Xerror_file	エラー・メッセージをファイルに出力します。
警告メッセージ出力制御	-Xno_warning	指定した警告メッセージの出力を抑止します。
サブコマンド・ファイル指定	@	サブコマンド・ファイルを指定します。

バージョン／ヘルプ表示指定

バージョン／ヘルプ表示指定オプションには，次のものがあります。

- -V
- -h

-V

asrh のバージョン情報を表示します。

[指定形式]

```
-V
```

- 省略時解釈
  asrh のバージョン情報を表示せずに，アセンブルを行います。

[詳細説明]

- asrh のバージョン情報を標準エラー出力に出力します。
  アセンブルは行いません。

[使用例]

- asrh のバージョン情報を標準エラー出力に出力します。

```
>asrh -V -Xcommon=v850e3v5
```

-h

asrh のオプションの説明を表示します。

[指定形式]

```
-h
```

- 省略時解釈
  asrh のオプションの説明を表示しません。

[詳細説明]

- asrh のオプションの説明を標準エラー出力に出力します。
  アセンブルは行いません。

[使用例]

- asrh のオプションの説明を標準エラー出力に出力します。

```
>asrh -h -Xcommon=v850e3v5
```

出力ファイル指定

出力ファイル指定オプションには，次のものがあります。

- -o
- -Xobj_path
- -Xprn_path

-o

出力ファイル名を指定します。

[指定形式]

```
-ofile
```

- 省略時解釈
  カレント・フォルダにファイルを出力します。
  出力オブジェクト・ファイル名は，ソース・ファイル名の拡張子を ".obj" に置き換えたものとなります。

[詳細説明]

- 出力オブジェクト・ファイル名を *file* に指定します。
- *file* がすでに存在する場合は，そのファイルを上書きします。
- 本 オプションを指定しても，フェイタル・エラーがある場合には，オブジェクト・ファイルの出力は行いません。
- 出力ファイルが複数の場合は，エラーとなります。
- *file* を省略した場合は，エラーとなります。

[使用例]

- オブジェクト・ファイルをファイル名 sample.obj で出力します。

```
>asrh -osample.obj -Xcommon=v850e3v5 main.asm
```

-Xobj_path

アセンブル途中に生成されるオブジェクト・ファイルを保存するフォルダを指定します。

[指定形式]

```
-Xobj_path[=path]
```

- 省略時解釈
  カレント・フォルダに，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。

[詳細説明]

- アセンブル途中に生成されるオブジェクト・ファイルを保存するフォルダを *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。
  存在しないフォルダ名を指定した場合は，エラーとなります。
- *path* には存在するファイル名を指定することも可能です。
  出力するオブジェクト・ファイルが 1 つの場合は，*path* というファイル名で保存します。
  出力するオブジェクト・ファイルが複数の場合は，エラーとなります。
  存在しないファイル名を指定した場合は，エラーとなります。
- =*path* を省略した場合は，カレント・フォルダに，ソース・ファイル名の拡張子を “.obj” で置き換えたファイル名でオブジェクト・ファイルを保存します。
- ソース・ファイルとして同じ名前のファイル（異なるフォルダにある場合を含む）を複数指定した場合は，警告を出力して，最後に指定したソース・ファイルに対するオブジェクト・ファイルのみを保存します。

[使用例]

- アセンブル途中に生成されるオブジェクト・ファイルをフォルダ D:¥sample に保存します。

```
>asrh -Xobj_path=D:¥sample -Xcommon=v850e3v5 main.asm
```

-Xprn_path

アセンブル・リスト・ファイルを保存するフォルダを指定します。

[指定形式]

```
-Xprn_path[=path]
```

- 省略時解釈
  アセンブル・リスト・ファイルを出力しません。

[詳細説明]

- アセンブル時に出力するアセンブル・リスト・ファイルを保存するフォルダを *path* に指定します。
- *path* に存在するフォルダ名を指定した場合は，フォルダ *path* に，ソース・ファイル名の拡張子を “.prn” で置き換えたファイル名でアセンブル・リスト・ファイルを保存します。
  存在しないフォルダ名を指定した場合は，エラーとなります。
- *path* には存在するファイル名を指定することも可能です。
- *path* というファイル名でアセンブル・リスト・ファイルを保存します。
  存在しないファイル名を指定した場合は，エラーとなります。
- =*path* を省略した場合は，カレント・フォルダに，ソース・ファイル名の拡張子を “.prn” で置き換えたファイル名でアセンブル・リスト・ファイルを保存します。

[使用例]

- アセンブル時に出力するアセンブル・リスト・ファイルをフォルダ D:¥sample に保存します。

```
>asrh -Xprn_path=D:¥sample -Xcommon=v850e3v5 main.asm
```

ソース・デバッグ制御

ソース・デバッグ制御オプションには，次のものがあります。

- -g

-g

ソース・デバッグ用の情報を出力します。

[指定形式]

```
-g
```

- 省略時解釈
  ソース・デバッグ用の情報を出力しません。

[詳細説明]

- ソース・デバッグ用の情報を出力ファイル中に出力します。
- 本オプションを指定することにより，ソース・デバッグが可能となります。

[使用例]

- ソース・デバッグ用の情報を出力ファイル中に出力します。

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

デバイス指定

デバイス指定オプションには，次のものがあります。

- -Xcommon
- -Xcpu

-Xcommon

デバイス共通のオブジェクト・ファイルを生成することを指定します。

[指定形式]

```
-Xcommon=series
```

- 省略時解釈
  エラーとなります。
  ただし，-V/-h オプションを指定している場合は，エラーとなりません。

[詳細説明]

- デバイス共通のオブジェクト・ファイルを生成することを指定します。
- 本オプションを指定した場合，ターゲットの命令セット・アーキテクチャの命令のみを使用し，また，命令セット・アーキテクチャに対応した共通のマジック・ナンバ *series* をオブジェクト・ファイルに埋め込みます。
- *series* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

v850e3v5	命令セット・アーキテクチャが V850E3V5 である品種をターゲット・デバイスとして指定したものとのリンクが可能です。
rh850	“v850e3v5” を指定した場合と同じです。

- *series* を省略した場合は，エラーとなります。

[使用例]

- 生成するオブジェクト・ファイルに，命令セット・アーキテクチャが V850E3V5 である品種に共通のマジック・ナンバを埋め込みます。

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

-Xcpu

指定したコア向けのオブジェクトを生成することを指定します。

[指定形式]

```
-Xcpu=core
```

- 省略時解釈
  -Xcommon オプションを指定した場合，G3M 向けオブジェクトを生成します。

[詳細説明]

- コア *core* 向けのオブジェクトを生成することを指定します。
- *core* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

g3m	G3M 向けオブジェクトを生成します。
g3k	G3K 向けオブジェクトを生成します。

- *core* を省略した場合は，エラーとなります。

[使用例]

- G3M 向けオブジェクトを生成することを指定します。

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

プリプロセッサ制御

プリプロセッサ制御オプションには，次のものがあります。

- -D
- -U
- -I

-D

アセンブラ・シンボルを定義します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- アセンブラ・シンボルとして *name* を定義します。
- アセンブリ・ソース・プログラムの前に，*name* .SET *def* を記述するのと同等です。
- *name* が，アセンブラ・シンボルには使用可能であるがプリプロセッサ・マクロには使用できない文字（@，.，~）を含む場合，警告を出力してアセンブラ・シンボルとしてのみ定義します。
- *name* を省略した場合は，エラーとなります。
- =*def* を省略した場合，*def* は 1 とみなします。
- 本オプションは，複数指定が可能です。
- 同じアセンブラ・シンボルに対して，本オプションと -U オプションを同時に指定した場合は，あとから指定したものが有効となります。

[使用例]

- アセンブラ・シンボルとして sample=256 を定義します。

```
>asrh -Dsample=256 -Xcommon=v850e3v5 main.asm
```

-U

-D オプションによるアセンブラ・シンボルの定義を解除します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- -D オプションによるアセンブラ・シンボル *name* の定義を解除します。
- *name* を省略した場合は，エラーとなります。
- 本オプションでは，*name* .SET *def* の記述による定義は解除できません。
- 本オプションは，複数指定が可能です。
- 同じアセンブラ・シンボルに対して，本オプションと -D オプションを同時に指定した場合は，あとから指定したものが有効となります。

[使用例]

- -D オプションによるアセンブラ・シンボル test の定義を解除します。

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

-I

インクルード・ファイルを検索するフォルダを指定します。

[指定形式]

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

- 省略時解釈
  インクルード・ファイルを 標準インクルード・ファイル・フォルダからのみ検索します。

[詳細説明]

- アセンブラ制御命令 $INCLUDE/$BINCLUDE で読み込むインクルード・ファイルを検索するフォルダを *path* に指定します。
  インクルード・ファイルの検索は，以下の順番で行います。

(1) -I オプションで指定したフォルダ

(2) ソース・ファイルのあるフォルダ

(3) 元の C ソース・ファイルのあるフォルダ

(4) カレント・フォルダ

- *path* が存在しない場合は，警告を出力します。
- *path* を省略した場合は，エラーとなります。

[使用例]

- インクルード・ファイルをフォルダ D:¥include，D:¥src，カレント・フォルダの順で検索します。

```
>asrh -ID:¥include -Xcommon=v850e3v5 D:¥src¥main.asm
```

日本語／中国語文字列制御

日本語／中国語文字列制御オプションには，次のものがあります。

- -Xcharacter_set

-Xcharacter_set

日本語／中国語の文字コードを指定します。

[指定形式]

```
-Xcharacter_set=code
```

- 省略時解釈
  日本語の文字コードを SJIS として扱います。

[詳細説明]

- ソース・ファイル中の日本語／中国語のコメント，文字列に対して，使用する文字コードを指定します。
- *code* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。
  なお，ソース・ファイル中で使用している文字コードと異なるものを指定した場合，動作は保証されません。

none	日本語／中国語の文字コードを解釈しません
euc_jp	EUC（日本語）
sjis	SJIS
utf8	UTF-8
big5	繁体字中国語
gb2312	簡体字中国語

- *code* を省略した場合は，エラーとなります。

[使用例]

- ソース・ファイル中の日本語のコメント，文字列に対して，使用する文字コードに EUC を指定します。

```
>asrh -Xcharacter_set=euc_jp -Xcommon=v850e3v5 main.asm
```

生成コード制御

生成コード制御オプションには，次のものがあります。

- -Xreg_mode
- -Xreserve_r2
- -Xep

-Xreg_mode

レジスタ・モードを指定します。

[指定形式]

```
-Xreg_mode=mode
```

- 省略時解釈
  32 レジスタ・モードのオブジェクト・ファイルを生成します。

[詳細説明]

- 指定したレジスタ・モードのオブジェクト・ファイルを生成します。
- ccrh が使用するレジスタを 32 本（32 レジスタ・モード），22 本（22 レジスタ・モード，または common レジスタ・モード）のいずれかに制限し，オブジェクト・ファイル内にレジスタ・モードを示すマジック・ナンバを埋め込みます。
- common レジスタ・モードは，レジスタ・モードに依存しないオブジェクト・ファイルを生成するために使用します。
- *mode* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

レジスタ・モード（*mode*）	作業用レジスタ	レジスタ変数用レジスタ
common	r10 ～ r14	r25 ～ r29
22	r10 ～ r14	r25 ～ r29
32	r10 ～ r19	r20 ～ r29

- *mode* を省略した場合は，エラーとなります。
- アセンブリ・ソース・ファイルに対して，使用可能なレジスタ以外のレジスタが使用されている場合は，エラーとなります。
- 本オプションは，すべてのソース・ファイルに対して同じレジスタ・モードを指定する必要があります。
  ソース・ファイルごとに指定を変えることはできません。
  レジスタ・モードが異なるオブジェクト・ファイルが混在している場合は，リンク時にエラーとなります。
- 本オプションを指定することにより，ソフトウエア・レジスタ・バンク機能のレジスタ・モードを切り替えることができます。

[使用例]

- 22 レジスタ・モードのオブジェクト・ファイルを生成します。

```
>asrh -Xreg_mode=22 -Xcommon=v850e3v5 main.asm
```

-Xreserve_r2

r2 レジスタを予約します。

[指定形式]

```
-Xreserve_r2
```

- 省略時解釈
  r2 レジスタの予約を行わずに，コンパイラで使用します。

[詳細説明]

- r2 レジスタを予約し，コンパイラでは r2 レジスタを使用しないコードを生成します。

[使用例]

- r2 レジスタを予約し，コンパイラでは r2 レジスタを使用しないコードを生成します。

```
>asrh -Xreserve_r2 -Xcommon=v850e3v5 main.asm
```

-Xep

ep レジスタの扱い方を指定します。

[指定形式]

```
-Xep=mode
```

- 省略時解釈
  ep レジスタを関数呼び出し前後で値を保証するレジスタとして扱います。

[詳細説明]

- ep レジスタの扱い方を指定します。
- *mode* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

fix	ep レジスタの値をプロジェクト全体で固定します。 プロジェクト内の #pragma section 指令において，以下の属性指定文字を使用した場合は，本パラメータを指定してください。 ep_auto，ep_disp4，ep_disp5，ep_disp7，ep_disp8，ep_disp16，ep_disp23
callee	ep レジスタを関数呼び出し前後で値を保証するレジスタとして扱います。 -Omap，または -Osmap オプションを指定した場合は，本パラメータを指定してください。

- *mode* を省略した場合は，エラーとなります。

[使用例]

- ep レジスタの値をプロジェクト全体で固定します。

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

アセンブラ制御指定

アセンブラ制御指定オプションには，次のものがあります。

- -Xasm_far_jump

-Xasm_far_jump

アセンブリ・ソース・ファイルに対して，far jump の出力を制御します。

[指定形式]

```
-Xasm_far_jump
```

- 省略時解釈
  jarl，または jr 命令としてアセンブルを行います。

[詳細説明]

- アセンブリ・ソース・ファイルに対して，ソース中に記述されたすべての jarl，および jr 命令を jarl32，および jr32 命令とみなしてアセンブルを行います。
- 個別の命令ごとに制御したい場合は，ソース中で jarl22/jarl32，jr22/jr32 のように明記します。
- 本オプションは，jump 命令には影響しません。
- 本オプションは，C ソース・ファイルに対しては無効です。

[使用例]

- アセンブリ・ソース中に記述されたすべての jarl，および jr 命令を jarl32，および jr32 命令とみなしてアセンブルを行います。

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

エラー出力制御

エラー出力制御オプションには，次のものがあります。

- -Xerror_file

-Xerror_file

エラー・メッセージをファイルに出力します。

[指定形式]

```
-Xerror_file=file
```

- 省略時解釈
  エラー・メッセージを標準エラー出力のみに出力します。

[詳細説明]

- エラー・メッセージを標準エラー出力，およびファイル *file* に出力します。
- *file* がすでに存在する場合は，そのファイルを上書きします。
- *file* を省略した場合は，エラーとなります。

[使用例]

- エラー・メッセージを標準エラー出力，およびファイル err に出力します。

```
>asrh -Xerror_file=err -Xcommon=v850e3v5 main.asm
```

警告メッセージ出力制御

警告メッセージ出力制御オプションには，次のものがあります。

- -Xno_warning

-Xno_warning

指定した警告メッセージの出力を抑止します。

[指定形式]

```
-Xno_warning=num[,num]...
-Xno_warning=num1-num2
```

- 省略時解釈
  すべての警告メッセージを出力します。

[詳細説明]

- 指定した警告メッセージの出力を抑止します。
- *num*，*num1*，*num2* には，メッセージ番号を指定します。
  存在しないメッセージ番号を指定した場合は，無視します。
- *num*，または *num1*，および *num2* を省略した場合は，エラーとなります。
- *num1-num2* の形式で指定すると，その範囲に含まれるメッセージ番号を指定したものとみなします。
- 本オプションで指定するメッセージ番号は，W に続く 7 桁の数字のうち，下位 5 桁です。
  メッセージ番号については，「CS+ 統合開発環境 ユーザーズマニュアル メッセージ編」を参照してください。
- 本オプションで制御することができるのは，コンパイラ，アセンブラの警告メッセージです（最適化リンカの警告メッセージは対象外です）。

[使用例]

- 警告メッセージ W0550002，W0550003 の出力を抑止します。

```
>asrh -Xno_warning=50002,50003 -Xcommon=v850e3v5 main.asm
```

サブコマンド・ファイル指定

サブコマンド・ファイル指定オプションには，次のものがあります。

- @

@

サブコマンド・ファイルを指定します。

[指定形式]

```
@file
```

- 省略時解釈
  コマンド・ラインで指定したオプション，およびファイル名のみを認識します。

[詳細説明]

- *file* をサブコマンド・ファイルとして扱います。
- *file* が存在しない場合は，エラーとなります。
- *file* を省略した場合は，エラーとなります。
- サブコマンド・ファイルについての詳細は，「2.4.2 サブコマンド・ファイルの使用方法」を参照してください。

[使用例]

- command.txt をサブコマンド・ファイルとして扱います。

```
>asrh @command.txt -Xcommon=v850e3v5
```

2.5.3 リンク・オプション

ここでは，リンク・フェーズのオプションについて説明します。

オプションに関する注意事項を以下に示します。

- オプションの大文字／小文字は区別しません。
- オプション，およびパラメータの大文字は，短縮形の指定が可能であることを表しています。
  大文字以降の文字の指定は任意です。

  例　-FOrm=Absolute の場合，例えば以下のように指定することも可能です。
  -fo=a
  -fo=abs
  -for=absolu

- パラメータとしてファイル名を指定する場合，“(”，および “)” を使用することはできません。
- ccrh コマンドに対して，リンク・オプションを指定する場合は，-Xlk_option オプションを使用する必要があります。

オプションの分類と説明を以下に示します。

表 2.4　リンク・オプション

分類	オプション	説明
入力制御	-Input	入力ファイルを指定します。
	-LIBrary	入力ライブラリ・ファイルを指定します。
	-Binary	入力バイナリ・ファイルを指定します。
	-DEFine	未定義シンボルを強制定義します。
	-ENTry	実行開始アドレスを指定します。
出力制御	-FOrm	出力形式を指定します。
	-DEBug	出力ファイル中にデバッグ情報を出力します。
	-NODEBug	デバッグ情報を出力しません。
	-RECord	出力するデータ・レコードのサイズを指定します。
	-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充てんします。
	-Message	インフォメーション・メッセージを出力します。
	-NOMessage	インフォメーション・メッセージの出力を抑止します。
	-MSg_unused	参照されない外部定義シンボルをユーザに通知します。
	-BYte_count	データ・レコードのバイト数の最大値を指定します。
	-PADDING	セクションの終端にデータを埋め込みます。
	-OVERRUN_FETCH	オーバーラン・フェッチに伴う空き領域の読み出しを回避します。
リスト出力	-LISt	リスト・ファイルを出力します。
	-SHow	リスト・ファイルへの出力情報を指定します。

分類	オプション	説明
セクション指定	-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シンボル・アドレス・ファイルに出力します。
	-ALIGNED_SECTION	セクションのアライメント数を 16 バイトに変更します。
ベリファイ指定	-CPu	セクションの割り付けアドレスの整合性をチェックします。
サブコマンド・ファイル指定	-SUbcommand	オプションをサブコマンド・ファイルで指定します。
その他	-S9	S9 レコードを終端に出力します。
	-STACk	スタック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指定します。
	-REName	外部シンボル名，セクション名を変更します。
	-DELete	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。
	-REPlace	ライブラリ・モジュールを置換します。
	-EXTract	ライブラリ・モジュールを抽出します。
	-STRip	ロード・モジュール・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	インフォメーション，ワーニング，エラーのメッセージ種別を変更します。
	-Hide	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。
	-Total_size	リンク後の合計セクション・サイズを標準エラー出力に表示します。
	-LOgo	コピーライトを出力します。
	-NOLOgo	コピーライトの出力を抑止します。
	-END	本オプション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。

入力制御

入力制御オプションには，次のものがあります。

- -Input
- -LIBrary
- -Binary
- -DEFine
- -ENTry

-Input

入力ファイルを指定します。

[指定形式]

```
-Input=file[(module[,module]...)][{,|△}file[(module[,module]...)]]...
```

- 省略時解釈
  なし

[詳細説明]

- 入力ファイル *file* を指定します。
  複数指定する場合は，カンマ（,），または空白文字で区切ります。
- ワイルドカード（*，?）も使用可能です。
  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。
  数字と英文字では数字を先に，英大文字と英小文字では英大文字を先に展開します。
- 入力ファイルとして指定できるのは，コンパイラ，またはアセンブラが出力したオブジェクト・ファイル，最適化リンカが出力したリロケータブル・ファイル，ロード・モジュール・ファイル，インテル拡張ヘキサ・ファイル，およびモトローラ・S タイプ・ファイルです。
  また，“*library*(*module*)” の形式で，ライブラリ内モジュールを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とき，モジュール名の指定がない場合は “.obj”，モジュール名の指定がある場合は “.lib” を指定したものとみなします。

[注意]

- 本オプションは，サブコマンド・ファイル内のみで使用することができます。
  本オプションをコマンド・ライン上で指定した場合は，エラーとなります。
  コマンド・ライン上で入力ファイルを指定する場合は，-input オプションなしで指定してください。

[使用例]

- a.obj と lib1.lib 内のモジュール e を入力します。
  ＜コマンド・ライン＞

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

＜サブコマンド・ファイル sub.txt ＞

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

- “c” で始まり，拡張子が “.obj” であるファイルをすべて入力します。
  ＜コマンド・ライン＞

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

＜サブコマンド・ファイル sub.txt ＞

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

［備考］

- 本オプションは，-form=object，および -extract オプションを指定した場合は無効となります。
- 入力ファイルにインテル拡張ヘキサ・ファイルを指定した場合は -form=hexadecimal オプション，モトローラ・S タイプ・ファイルを指定した場合は -form=stype オプションのみを指定することができます。
  出力ファイル名を指定していない場合は，“ *先頭入力ファイル名* _combine. *拡張子* ” となります（入力ファイルが a.mot の場合，出力ファイルは a_combine.mot となります）。

-LIBrary

入力ライブラリ・ファイルを指定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- 入力ライブラリ・ファイル *file* を指定します。
  複数指定する場合は，カンマ（,）で区切ります。
- ワイルドカード（*，?）も使用可能です。
  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。
  数字と英文字では数字を先に，英大文字と英小文字では英大文字を先に展開します。
- 入力ファイル名に拡張子の指定がない場合は “.lib” を指定したものとみなします。
- -form=library，または -extract オプションと同時に指定した場合は，指定したライブラリ・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ク処理後に，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，以下の順序で行います。
  - 本オプションで指定したユーザ・ライブラリ・ファイル（指定順）
  - 本オプションで指定したシステム・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，HLNK_LIBRARY2，HLNK_LIBRARY3 注の順）

  注 環境変数についての詳細は，「2.3 環境変数」を参照してください。

[使用例]

- a.lib と b.lib を入力します。

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

- “c” で始まり，拡張子が “.lib” であるファイルをすべて入力します。

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

-Binary

入力バイナリ・ファイルを指定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- 入力バイナリ・ファイル *file* を指定します。
  複数指定する場合は，カンマ（,）で区切ります。
- 入力ファイル名に拡張子の指定がない場合は “.bin” を指定したものとみなします。
- 入力したバイナリ・データは，指定したセクション *section* 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。
  section を省略した場合は，エラーとなります。
- シンボル *symbol* を指定すると，定義シンボルとしてリンクすることもできます。
  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の先頭に “_” 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 *attribute* に指定可能なものは，CODE，または DATA です。
  attribute を省略した場合は，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 *alignment* に指定可能な値は 2 の累乗（1，2，4，8，16，32）です。
  それ以外の値を指定することはできません。
  alignment を省略した場合は，デフォルトとして，1 が有効となります。

[使用例]

- 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。

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

[備考]

- 本オプションは，-form={object|library} オプション，または -strip オプションを指定した場合は無効となります。
- 入力オブジェクト・ファイルを指定していない場合，本オプションは指定することができません。

-DEFine

未定義シンボルを強制定義します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- 未定義シンボル *symbol1* を外部定義シンボル *symbol2*，または数値 *value* で強制定義します。
- *value* 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解釈します。
  先頭が 0 の場合は，常に数値と解釈します。
- シンボル名が C 変数名の場合は，プログラム中での定義名の先頭に “_” を付加します。

[使用例]

- _sym1 を外部定義シンボル data と同値として定義します。

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

- _sym2 を 0x4000 として定義します。

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

[備考]

- 本オプションは，-form={object|relocate|library} オプションを指定した場合は無効となります。

-ENTry

実行開始アドレスを指定します。

[指定形式]

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

- 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 *symbol*，またはアドレス *address* で指定します。
- *address* 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解釈します。
  先頭が 0 の場合は，常にアドレスと解釈します。
- シンボル名が C 変数名の場合は，プログラム中での定義名の先頭に “_” を付加します。

[使用例]

- C の main 関数を実行開始アドレスとして指定します。

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

- 0x100 を実行開始アドレスとして指定します。

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

[備考]

- 本オプションは，-form={object|relocate|library} オプション，または -strip オプションを指定した場合は無効となります。

出力制御

出力制御オプションには，次のものがあります。

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

-FOrm

出力形式を指定します。

[指定形式]

```
-FOrm={format}
```

- 省略時解釈
  ロード・モジュール・ファイルを出力します（-form=absolute オプションの指定と同じです）。

[詳細説明]

- 出力形式 *format* を指定します。
- *format* に指定可能なものを以下に示します。

Absolute	ロード・モジュール・ファイルを出力します。
Relocate	リロケータブル・ファイルを出力します。
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。
Library[={S|U}]	ライブラリ・ファイルを出力します。 library=s を指定した場合は，出力ファイルをシステム・ライブラリ・ファイルとします。 library=u を指定した場合は，出力ファイルをユーザ・ライブラリ・ファイルとします。 library のみを指定した場合は，library=u を指定したものとみなします。
Hexadecimal	インテル拡張ヘキサ・ファイルを出力します。 詳細については，「3.5 インテル拡張ヘキサ・ファイル」を参照してください。
Stype	モトローラ・S タイプ・ファイルを出力します。 詳細については，「3.6 モトローラ・S タイプ・ファイル」を参照してください。
Binary	バイナリ・ファイルを出力します。

- *format* を省略した場合は，Absolute を指定したものとみなします。

[備考]

- 出力形式と入力ファイル，他のオプションとの関係を以下に示します。

図 2.2 出力形式と入力ファイル，他のオプションとの関係

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]
Absolute	-strip あり	ロード・モジュール・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，-cpu，-start，-rom，-entry，-output，-map，-hide，-compress，-rename，-delete，-define，-fsymbol，-stack，-memory， -msg_unused，-show={symbol|reference|xreference}，-aligned_section，-overrun_fetch

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
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, -show={symbol|xreference}
Object	-extract あり	ライブラリ・ファイル	-library, -output
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, -rename, -delete, -define, -fsymbol, -stack, -record, -s9 注 2, -byte_count 注 3, -memory, -msg_unused, -show={symbol|reference|xreference}, -aligned_section, -overrun_fetch 注 4
		ロード・モジュール・ファイル	-input, -output, -record, -s9 注 2, -byte_count 注 3, -show={symbol|reference|xreference}
		インテル拡張ヘキサ・ファイル注 5	-input, -output
		モトローラ・S タイプ・ファイル注 5	-input, -output, -s9 注 2
Library	-strip あり	ライブラリ・ファイル	-library, -output, -memory 注 6, -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, -memory 注 6, -show={symbol|section}

注 1. 以下のオプションは，常に指定可能です。
-message, -nomessage, -change_message, -logo, -nologo, -form, -list, -subcommand

注 2. -s9 オプションは，-form=stype オプションを指定した場合のみ指定可能です。

注 3. -byte_count オプションは，-form= hexadecimal オプションを指定した場合のみ指定可能です。

注 4. -overrun_fetch オプションは，-form=hexadecimal，または -form=stype オプションを指定した場合のみ指定可能です。

注 5. 入力ファイルにインテル拡張ヘキサ・ファイルを指定した場合は -form=hexadecimal オプション，モトローラ・S タイプ・ファイルを指定した場合は -form=stype オプションのみを指定することができます。

注 6. -memory オプションは，-hide オプションを指定した場合は指定することはできません。

[使用例]

- a.obj，b.obj からリロケータブル・ファイル c.rel を出力します。

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

- lib.lib からモジュール a を取り出し，オブジェクト・ファイルとして出力します。

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

- lib.lib からモジュール a を取り出し，ライブラリ・ファイル exta.lib を出力します。

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

- lib.lib からモジュール a を取り出し，リロケータブル・ファイル a.rel を出力します。

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

-DEBug

出力ファイル中にデバッグ情報を出力します。

[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（-debug オプションの指定と同じです）。

[詳細説明]

- 出力ファイル中にデバッグ情報を出力します。

[使用例]

- 出力ファイル中にデバッグ情報を出力します。

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

[備考]

- 本オプションは，-form={object|library|hexadecimal|stype|binary}，-strip オプション，または -extract オプションを指定した場合は無効となります。
- -form=absolute オプション，および -output オプションで出力ファイル名を複数した場合は，デバッグ情報を出力しません。

-NODEBug

デバッグ情報を出力しません。

[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（-debug オプションの指定と同じです）。

[詳細説明]

- デバッグ情報を出力しません。

[使用例]

- デバッグ情報を出力しません。

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

[備考]

- 本オプションは，-form={object|library|hexadecimal|stype|binary}，-strip オプション，または -extract オプションを指定した場合は無効となります。

-RECord

出力するデータ・レコードのサイズを指定します。

[指定形式]

```
-RECord={record}
```

- 省略時解釈
  それぞれのアドレスにあわせて，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 *record* で出力します。
- *record* に指定可能なものを以下に示します。

H16	ヘキサ・レコード
H20	拡張ヘキサ・レコード
H32	32 ビット・ヘキサ・レコード
S1	S1 レコード
S2	S2 レコード
S3	S3 レコード

- 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に，アドレスに合わせてデータ・レコードを選択します。

[使用例]

- アドレス範囲に関係なく，32 ビット・ヘキサ・レコードで出力します。

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

[備考]

- 本オプションは，-form={hexadecimal|stype} オプションを指定していない場合は無効となります。
- -form=hexadecimal オプション指定時に -record={S1|S2|S3} オプションを指定した場合，および，-form=stype オプション指定時に -record={H16|H20|H32} オプションを指定した場合は，エラーとなります。

-ROm

ROM から RAM へマップするセクションを指定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- 初期値ありデータ領域の 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でリロケーションします。
- ROM セクション *ROMsection* 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 *RAMsection* 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。

[使用例]

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

```
>rlink a.obj b.obj -rom=.data=.data.R -start=.data/100,.data.R/8000
```

[備考]

- 本オプションは，-form={object|relocate|library} オプション，または -strip オプションを指定した場合は無効となります。

-OUtput

出力ファイルを指定します。

[指定形式]

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

- 省略時解釈
  出力ファイル名を「*先頭入力ファイル名*. *デフォルト拡張子*」とします。
  デフォルト拡張子を以下に示します。

-form=absolute オプション指定時	:	abs
-form=relocate オプション指定時	:	rel
-form=object オプション指定時	:	obj
-form=library オプション指定時	:	lib
-form=hexadecimal オプション指定時	:	hex
-form=stype オプション指定時	:	mot
-form=binary オプション指定時	:	bin

[詳細説明]

- 出力ファイル *file* を指定します。
- *address1*，*address2* には，出力範囲の先頭アドレス，終了アドレスを 16 進数で指定します。
  出力範囲に "-" を指定した場合は常にアドレスと解釈します。
- *section* には，出力するセクションを指定します。
  複数指定する場合は，コロン（:）で区切ります。
- -form={absolute|hexadecimal|stype|binary} オプションと同時に指定した場合は，複数のファイルを指定することができます。

[使用例]

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

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

- 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。

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

[備考]

- 入力ファイルがインテル拡張・ヘキサ・ファイル，またはモトローラ・S タイプ・ファイルの場合，本オプションで複数の出力ファイルを指定することはできません。
  本オプションを省略した場合，出力ファイル名は " *先頭入力ファイル名* _combine. *拡張子* " となります（入力ファイルが a.mot の場合，出力ファイルは a_combine.mot となります）。

-MAp

外部変数割り付け情報ファイルを出力します。

[指定形式]

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

- 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 *file* を出力します。
- ファイル名の指定を省略した場合は，-output オプションで指定したファイル名，または " *先頭入力ファイル名* .bls" というファイル名となります。
- 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異なる場合は，エラーとなります。

[使用例]

- 外部変数割り付け情報ファイル file.bls を出力します。

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

[備考]

- 本オプションは，-form={absolute|hexadecimal|stype|binary} オプションを指定した場合のみ有効となります。

-SPace

出力範囲のメモリの空き領域を充てんします。

[指定形式]

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

- 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 *data* で充てんします。
- *data* に指定可能なものを以下に示します。

数値	16 進数の数値
Random	乱数

- 空き領域の充てん方法は，-output オプションで指定する出力範囲によって，以下のように異なります。
  - 出力範囲としてセクションを指定した場合
  - 指定したセクション間に空き領域が存在した場合に，指定データを出力します。
  - 出力範囲として先頭アドレス，終了アドレスを指定した場合
  - 指定した範囲内に空き領域が存在した場合に，指定データを出力します。
- 出力データ・サイズは，1，2，4 バイト単位で有効となり，本オプションで指定する 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は，上位桁を 0 拡張し，4 バイトのデータとして扱います。
  奇数桁データを指定した場合は，上位桁に 0 拡張して，偶数桁入力として扱います。
- 空き領域のサイズが出力データ・サイズの倍数でない場合は，出力できるだけ出力し，警告を出力します。

[使用例]

- 100H 番地から 2FFH 番地の範囲内で，メモリの空き領域に対して ffH で充てんします。

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

[備考]

- 本オプションにてデータの指定を省略した場合は，空き領域への充てんを行いません。
- 本オプションは，-form={binary|stype|hexadecimal} オプションを指定した場合のみ有効となります。
- 本オプションは，-output オプションにて出力範囲を指定しなかった場合は無効となります。

-Message

インフォメーション・メッセージを出力します。

[指定形式]

```
-Message
```

- 省略時解釈
  インフォメーション・メッセージの出力を抑止します（-nomessage オプションの指定と同じです）。

[詳細説明]

- インフォメーション・メッセージを出力します。

[使用例]

- インフォメーション・メッセージを出力します。

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

-NOMessage

インフォメーション・メッセージの出力を抑止します。

[指定形式]

```
-NOMessage[=num[-num][,num[-num]]...]
```

- 省略時解釈
  インフォメーション・メッセージの出力を抑止します。

[詳細説明]

- インフォメーション・メッセージの出力を抑止します。
- メッセージ番号 *num* を指定すると，指定した番号のメッセージの出力を抑止します。
  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。
- *num* 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）。
- ワーニング，またはエラー種別のメッセージ番号を指定した場合は，-change_message オプションでインフォメーション種別に変更したと仮定し，メッセージの出力を抑止します。

[使用例]

- M0560004，M0560100 ～ M0560103，および M0560500 のメッセージの出力を抑止します。

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

-MSg_unused

参照されない外部定義シンボルをユーザに通知します。

[指定形式]

```
-MSg_unused
```

- 省略時解釈
  なし

[詳細説明]

- 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通知します。

[使用例]

- 参照されない外部定義シンボルをユーザに通知します。

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

[備考]

- 本オプションは，入力ファイルがロード・モジュール・ファイルの場合は無効となります。
- 本オプションは，-message オプションと同時に指定する必要があります。
- 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。
  その場合は，関数定義に static 宣言を追加することにより，メッセージ出力を抑止することができます。
- 同一ファイル内の定数シンボルへの参照がある場合は，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。

-BYte_count

データ・レコードのバイト数の最大値を指定します。

[指定形式]

```
-BYte_count=num
```

- 省略時解釈
  バイト数の最大値を FF として，インテル拡張ヘキサ・ファイルを生成します。

[詳細説明]

- インテル拡張ヘキサ・ファイルを生成する際に，データ・レコードのバイト数の最大値 *num* を指定します。
- *num* には，1 バイトの 16 進数（01 ～ FF）を指定します。

[使用例]

- データ・レコードのバイト数の最大値として 10 を指定します。

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

[備考]

- 本オプションは，生成ファイルがインテル拡張ヘキサ・ファイルでない場合（-form=hexadecimal オプション未指定時）は無効となります。

-PADDING

セクションの終端にデータを埋め込みます。

[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，セクションの終端にデータを埋め込みます。

[使用例]

- 以下の場合，P セクションに 2 バイトのパディング・データを埋め込み，サイズを 0x08 バイトにしてリンク処理を行います。
  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

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

- 以下の場合，P セクションに 2 バイトのパディング・データを埋め込み，サイズを 0x08 バイトにしてリンク処理を行うと，C セクションと重複してしまうため，エラーを出力します。
  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

```
>rlink a.obj b.obj -start=P/0,C/7 -padding
```

[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わないため，絶対アドレス・セクションのサイズはユーザにて調整してください。
- V1.00.01 ではテキストデータ，const 変数，および初期値がある変数のセクションのみを対象としてパディング・データを埋め込んでいましたが，V1.01.00 以降では初期値がない変数のセクションも対象とします。

-OVERRUN_FETCH

オーバーラン・フェッチに伴う空き領域の読み出しを回避します。

[指定形式]

```
-OVERRUN_FETCH
```

- 省略時解釈
  なし

[詳細説明]

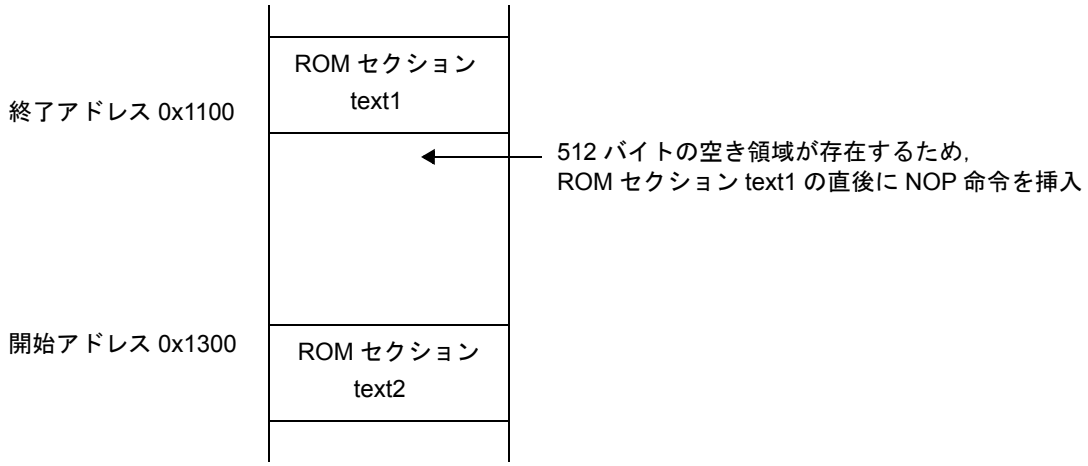
- セクション間，および終端コード・セクションの直後に 128 バイト以上の空き領域が存在する場合，空き領域に対して 128 バイトの NOP 命令を挿入します。
  NOP 命令の挿入は -start オプションの配置に従って挿入し，挿入する直前のセクション名に対して以下のセクション名で出力します。

```
$sss_fetch??
```

sss ： 挿入する直前のセクション名
?? ： 01 ～ 99

(1) セクション間への挿入

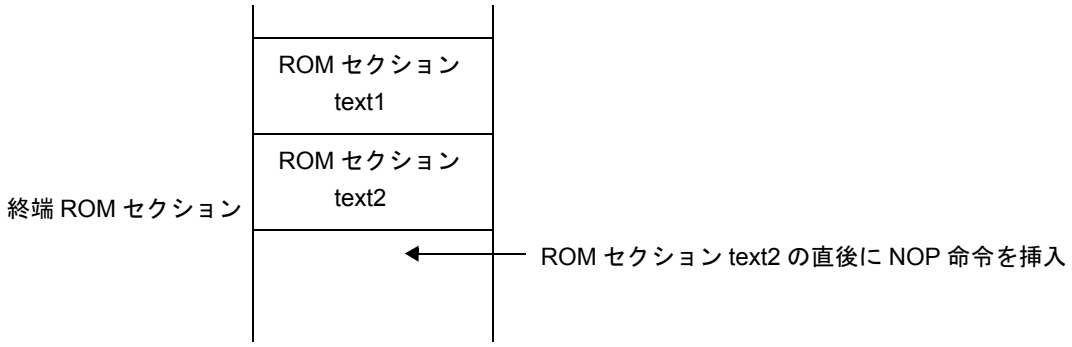
- ROM セクション（実行可能なプログラム・セクション，またはプログラム・セクション以外の ROM 領域配置セクション）と ROM セクション，および ROM セクションと RAM セクション（RAM 領域配置セクション）の間に 128 バイト以上の空き領域が存在する場合に NOP 命令を挿入します。



- ROM セクションと RAM セクションは，ROM セクションより大きいアドレスに RAM セクションが配置された場合に限り，NOP 命令を挿入します（ROM セクションより小さいアドレスに RAM セクションが配置された場合は NOP 命令を挿入しません）。

(2) 終端セクション直後への挿入

- 終端 ROM セクションの直後に NOP 命令を挿入します。



- -cpu オプションでアドレス範囲を指定する際，終端セクションの終端アドレスとして指定した終了アドレスの間に 128 バイトの空き領域がない場合，NOP 命令は挿入されません。
- 終端セクションの終端アドレスと 8M バイト境界のアドレスとの間に 128 バイトの空き領域がない場合，NOP 命令は挿入されません。

[使用例]

- オーバーラン・フェッチに伴う空き領域の読み出しを回避します。

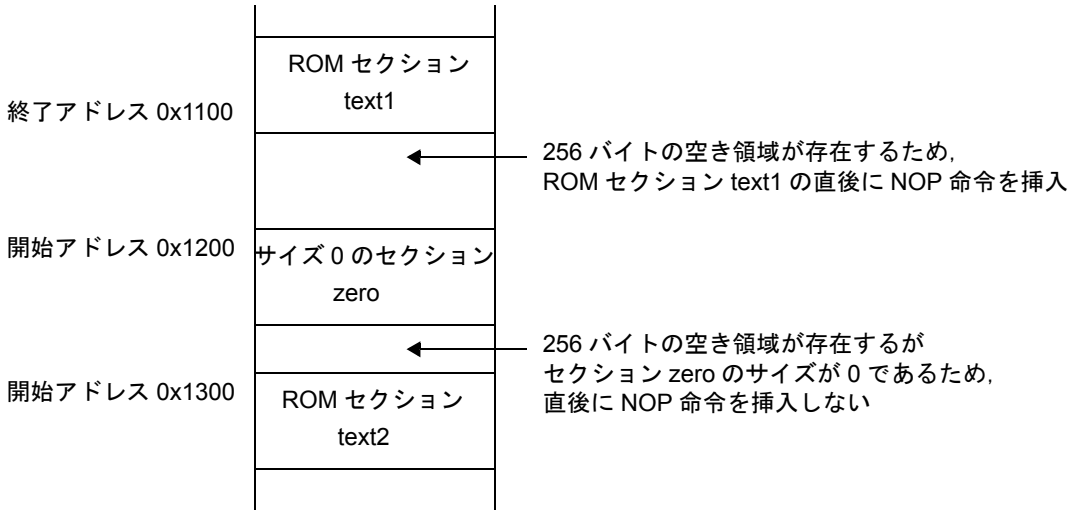
```
>rlink a.obj b.obj -overrun_fetch
```

[備考]

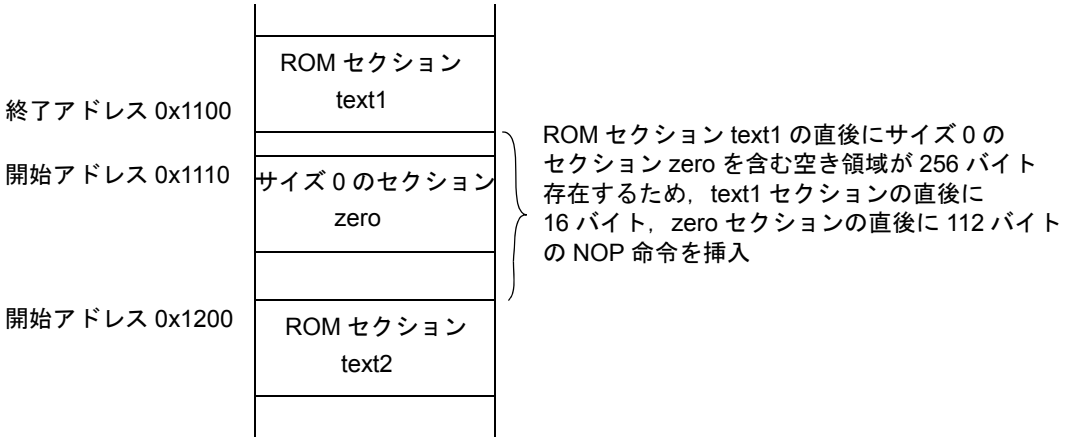
- -cpu オプションでアドレス範囲を指定していない場合，かつ終端セクションが ROM セクションの場合，終端 ROM セクションの直後に常に NOP 命令を挿入します。
- 本オプションは，入力ファイルがロード・モジュール・ファイルの場合は無効となります。
- 本オプションは，-form={object|relocate|library} オプションを指定した場合は無効となります。
- -start オプションでオーバーレイ配置指定を行った場合，オーバーレイ配置指定を含む start 列には NOP 命令の挿入を行いません。

例 1. -start=A,B,E/400,C,D:F:G/8000 オプションを指定した場合，オーバーレイ配置指定を含むセクション C，D，F，および G に対するセクション間，および終端コード・セクションに対して，NOP 命令の挿入を行いません。

例 2. サイズ 0 のセクションの直後に 128 バイト以上の空き領域が存在する場合，NOP 命令の挿入を行いません。



例 3. ROM セクションの直後に配置されたサイズ 0 のセクションを含む 128 バイト以上の空き領域が存在する場合，NOP 命令を挿入します。



リスト出力

リスト出力オプションには，次のものがあります。

- -LISt
- -SHow

-LISt

リスト・ファイルを出力します。

[指定形式]

```
-LISt[=file]
```

- 省略時解釈
  なし

[詳細説明]

- リスト・ファイル *file* を出力します。
- ファイル名の指定を省略した場合は，以下のリスト・ファイルを出力します。

指定オプション	ファイルの種類	ファイル名
-form=library オプション，または -extract オプション	ライブラリ・リスト・ファイル	*出力ファイル名*[注] .lbp
上記以外	リンク・マップ・ファイル	*出力ファイル名*[注] .map

注 出力ファイルが複数存在する場合は，先頭出力ファイル名となります。

[使用例]

- リンク・マップ・ファイルを file.map に出力します。

```
>rlink a.obj b.obj -list=file.map
```

-SHow

リスト・ファイルへの出力情報を指定します。

[指定形式]

```
-SHow[=info[,info]...]
```

- 省略時解釈
  なし

[詳細説明]

- リスト・ファイルへの出力情報 *info* を指定します。
- *info* に指定可能なものを以下に示します。

出力情報（*info*）	-form=library オプション指定時	-form=library オプション以外指定時
SYmbol	モジュール内シンボル名を出力	シンボル・アドレス，サイズ，種別，最適化内容を出力
Reference	指定不可	シンボル・アドレス，サイズ，種別，最適化内容，シンボルの参照回数を出力
SEction	モジュール内セクション名を出力	指定不可
Xreference	指定不可	クロス・リファレンス情報を出力
Total_size	指定不可	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出力
struct	指定不可	シンボル情報内にコンパイル時に -g を指定したファイル内で定義した構造体／共用体メンバ情報を出力します。（-form=rel もしくは obj を指定した場合は無効）
ALL	モジュール内シンボル名，セクション名を出力	-form=relocate オプション指定時 -show=symbol,xreference,total_size オプション指定時と同じ情報を出力 -form=absolute オプション指定時 -show=symbol,reference,xreference,total_size 指定時と同じ情報を出力 -form=hexadecimal/stype/binary オプション指定時 -show=symbol,reference,xreference,total_size 指定時と同じ情報を出力 -form=object オプション指定時 指定不可

備考　出力情報の詳細については，「3.2 リンク・マップ・ファイル」，「3.4 ライブラリ・リスト・ファイル」を参照してください。

- 出力情報の指定を省略した場合については，［備考］を参照してください。
- -extract オプションを指定した場合，出力情報（*info*）を指定することはできません。

[使用例]

- リンク・マップ・ファイルにシンボル・アドレス，サイズ，種別，最適化内容，シンボルの参照回数を出力します。

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

［備考］

- -form オプションと -show，または -show=all オプションの指定により，出力情報 *info* が有効／無効になる組み合わせを以下に示します。

		Symbol	Reference	Section	Xreference	Total_size
-form=absolute	-show のみ	有効	有効	無効	無効	無効
	-show=all	有効	有効	無効	有効	有効
-form=library	-show のみ	有効	無効	有効	無効	無効
	-show=all	有効	無効	有効	無効	無効
-form=relocate	-show のみ	有効	無効	無効	無効	無効
	-show=all	有効	無効	無効	有効[注 1]	有効
-form=object	-show のみ	有効	有効	無効	無効	無効
	-show=all	無効	無効	無効	無効	無効
-form=hexadecimal[注 2]/stype[注 3]/binary	-show のみ	有効	有効	無効	無効	無効
	-show=all	有効	有効	無効	有効	有効[注 1]

注 1. 入力ファイルがロード・モジュール・ファイルの場合は無効となります。

注 2. 入力ファイルがインテル拡張ヘキサ・ファイルの場合は，-show オプションを指定することはできません。

注 3. 入力ファイルがモトローラ・S タイプ・ファイルの場合は，-show オプションを指定することはできません。

なお，クロス・リファレンス情報の出力については以下の制限があります。

- 入力ファイルがロード・モジュール・ファイルの場合，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ついての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。
- -show=total_size オプションで表示する情報は，-total_size オプションにより表示する情報と同じです。

セクション指定

セクション指定オプションには，次のものがあります。

- -STARt
- -FSymbol
- -ALIGNED_SECTION

-STARt

セクションの開始アドレスを指定します。

[指定形式]

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

- 省略時解釈
  絶対アドレス・セクションをアドレスの小さい順に配置し，絶対アドレス・セクションの末尾から指定された入力ファイル順に出現する相対アドレス・セクションを配置します。

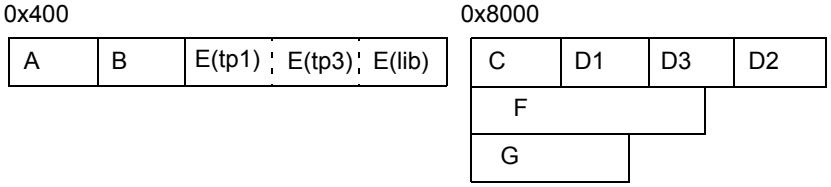
[詳細説明]

- セクション *section* の開始アドレス *address* を指定します。
  address は 16 進数で指定します。
- *section* にはワイルドカード（*）も使用可能です。
  ワイルドカードで指定したセクションは，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（カンマ（,）で区切って指定）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，丸かっこ “()” 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することができます。
- 同一セクション内のオブジェクトは，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。
- アドレスの指定を省略した場合は，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。

[使用例]

- 以下の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示します（かっこ内は各オブジェクトが持つセクションです）。
  tp1.obj(A,D1,E)
  tp2.obj(B,D3,F)
  tp3.obj(C,D2,E,G)
  lib.lib(E)

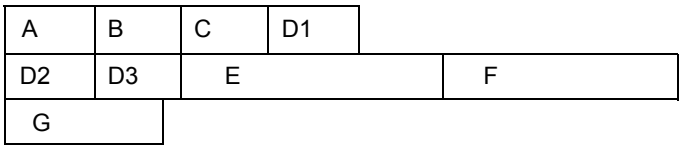
  - -start=A,B,E/400,C,D*:F:G/8000 オプションを指定した場合



  - “:” 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 オプションを指定した場合

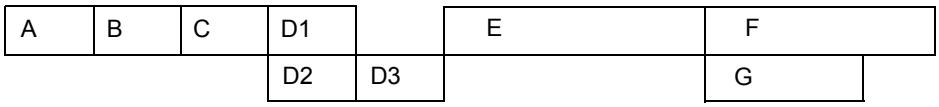
0x400



  - ":" 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，それぞれの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 オプションを指定した場合

0x400



  - "()" で同一アドレス配置を囲んだ場合，"()" 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（この例の場合は 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］

- 本オプションは，-form={object|relocate|library} オプション，または -strip オプションを指定した場合は無効となります。
- "()" は，ネストして記述することはできません。
- "()" 内では，1 つ以上のコロン ":" の記述が必要です。
  ":" を記述しない場合，"()" を記述することはできません。
- "()" を記述した場合，"()" の外にコロン ":" を記述することはできません。

-FSymbol

外部定義シンボルをシンボル・アドレス・ファイルに出力します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- セクション *section* 内の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（シンボル・アドレス・ファイル）に出力します。
  ファイル名は " *出力ファイル名* .fsy" となります。

[使用例]

- セクション sct2，sct3 内の外部定義シンボルをシンボル・アドレス・ファイル test.fsy に出力します。

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

シンボル・アドレス・ファイル test.fsy の出力例を以下に示します。

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

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

[備考]

- 本オプションは，-form={object|relocate|library} オプション，または -strip オプションを 指定した場合は無効となります。

-ALIGNED_SECTION

セクションのアライメント数を 16 バイト に変更します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- セクション *section* のアライメント数を 16 バイト に変更します。

[使用例]

- A セクションのアライメント数を 16 バイト に変更します。

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

[備考]

- 本オプションは，-form={object|relocate|library} オプション，および -extract，-strip オプションを 指定した場合は無効となります。

ベリファイ指定

ベリファイ指定オプションには，次のものがあります。

- -CPu

-CPu

セクションの割り付けアドレスの整合性をチェックします。

[指定形式]

```
-CPu=type=address1-address2[,type=address1-address2]...
```

- 省略時解釈
  なし

[詳細説明]

- セクションの割り付けアドレスの整合性をチェックします。
  メモリ種別 *type* のセクションの割り付けアドレスに対して，指定したアドレス範囲に収まらない場合は，エラーを出力します。
- *type* に指定可能なものを以下に示します。
  これ以外のものを指定した場合は，エラーとなります。

ROm	セクションの割り付け領域を ROM とします。
RAm	セクションの割り付け領域を RAM とします。
FIX	セクションの割り付け領域をアドレス固定の領域（I/O エリアなど）とします。 アドレス範囲が ROM，および RAM と重複した場合は，“FIX” を有効とします。

- *address1*，*address2* には，整合性をチェックするアドレス範囲の先頭アドレス，終了アドレスを 16 進数で指定します。

[使用例]

- .text セクションが 0x100 ～ 0x1FF，.bss セクションが 0x200 ～ 0x2FF の範囲に収まる場合は正常終了します。
  収まらない場合はエラーを出力します。

```
>rlink a.obj b.obj -start=.text/100,.bss/200 -cpu=ROM=100-1FF,RAM=200-2FF
```

[備考]

- 本オプションは，-form={object|relocate|library} オプション，または -strip オプションを指定した場合は無効となります。

サブコマンド・ファイル指定

サブコマンド・ファイル指定オプションには，次のものがあります。

- -SUbcommand

-SUbcommand

オプションをサブコマンド・ファイルで指定します。

[指定形式]

```
-SUbcommand=file
```

- 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 *file* で指定します。
- 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・ライン上の本オプション指定位置に展開し，実行します。
- サブコマンド・ファイルについての詳細は，「2.4.2 サブコマンド・ファイルの使用方法」を参照してください。

[使用例]

- サブコマンド・ファイル test.sub を以下の内容で作成します。

```
input file2.obj file3.obj        ; ここはコメントです。
library lib1.lib, &              ;“&” は継続行を示します。
lib2.lib
```

コマンド・ライン上でサブコマンド・ファイル test.sub を指定します。

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

コマンド・ラインは以下のように展開され，ファイルの入力順序は，file1.obj，file2.obj，file3.obj，file4.obj となります。

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

その他

その他のオプションには，次のものがあります。

- -S9
- -STACk
- -COmpress
- -NOCOmpress
- -MEMory
- -REName
- -DELete
- -REPlace
- -EXTract
- -STRip
- -CHange_message
- -Hide
- -Total_size
- -LOgo
- -NOLOgo
- -END
- -EXIt

-S9

S9 レコードを終端に出力します。

[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。

[使用例]

- 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。

```
>rlink a.obj b.obj -form=stype -s9
```

[備考]

- 本オプションは，-form=stype オプションを指定していない場合は無効となります。

-STACk

スタック情報ファイルを出力します。

[指定形式]

```
-STACk
```

- 省略時解釈
  なし

[詳細説明]

- スタック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” となります。

[使用例]

- スタック情報ファイル c.sni を出力します。

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

[備考]

- 本オプションは，-form={object|relocate|library} オプション，および -strip オプションを指定した場合は無効となります。

-COmpress

デバッグ情報を圧縮します。

[指定形式]

```
-COmpress
```

- 省略時解釈
  デバッグ情報を圧縮しません（-nocompress オプションの指定と同じです）。

[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。

[使用例]

- デバッグ情報を圧縮します。

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

[備考]

- 本オプションは，-form={object|relocate|library|hexadecimal|stype|binary} オプション，または -strip オプションを指定した場合は無効となります。

-NOCOmpress

デバッグ情報を圧縮しません。

[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。
- 本オプションを指定すると，-compress オプションを指定した場合に比べてリンク時間が短くなります。

[使用例]

- デバッグ情報を圧縮しません。

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

-MEMory

リンク時に使用するメモリ量を指定します。

[指定形式]

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

- 省略時解釈
  従来通りの処理を行います（-memory=high オプションの指定と同じです）。

[詳細説明]

- リンク時に使用するメモリ量 *occupancy* を指定します。
- *occupancy* に指定可能なものを以下に示します。

High	従来通りの処理を行います。
Low	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 High を指定した場合よりも処理が遅くなります。

- *occupancy* を省略した場合は，High を指定したものとみなします。
- 大規模なプロジェクトをリンクした際，最適化リンカのメモリ使用量が稼動マシンの実装メモリ量を超えてしまい，動作が遅くなっているような場合は，*occupancy* に Low を指定してください。

[使用例]

- 使用するメモリ量の削減を行います。

```
>rlink a.obj b.obj -memory=low
```

[備考]

- 以下の場合，-memory=low オプションの指定は無効となります。
  - -form={absolute|hexadecimal|stype|binary} オプションと以下のオプションを同時に指定した場合
    -compress，-delete，-rename，-map，-stack オプションのいずれか
    -list オプションと -show[={reference|xreference}] オプションを同時指定
  - -form=library オプションと以下のオプションを同時に指定した場合
    -delete，-rename，-extract，-hide，-replace オプションのいずれか
  - -form={object|relocate} オプションと以下のオプションを同時に指定した場合
    -extract オプション

  また，入力ファイルや出力ファイルの形式によっても無効となる組み合わせがあります。
  詳細については，「図 2.2 出力形式と入力ファイル，他のオプションとの関係」を参照してください。

-REName

外部シンボル名，セクション名を変更します。

[指定形式]

```
-REName={[file](name1=name2[,name1=name2]...)|[module](name1=name2[,name1=name2]...)}
[[,{[file](name1=name2[,name1=name2]...)|[module](name1=name2[,name1=name2]...)}]...]
```

- 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。
- *name1* には変更対象のシンボル名，またはセクション名，*name2* には変更後のシンボル名，またはセクション名を指定します。
- 特定のファイル *file*，または特定のライブラリ内モジュール *module* 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 変数名を指定する場合は，プログラム中での定義名の先頭に "_" を付加します。
- 関数名を変更した場合の動作を保証することはできません。
- 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は，シンボル名を優先します。
- 同一のファイル名，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。

[使用例]

- シンボル名 _sym1 を data に変更します。

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

- ライブラリ・モジュール lib1 内の .text セクションを P セクションに変更します。

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

[備考]

- 本オプションは，-extract オプション，または -strip オプションと同時に指定した場合は無効となります。
- -form=absolute オプションを指定した場合は，入力されたライブラリのセクション名を変更することはできません。
- コンパイル・オプション -Xmerge_files と本オプションを組み合わせて使用した場合，動作は保証されません。

-DELete

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。

[指定形式]

```
-DELete={[file](symbol[,symbol]...)|module}[,{[file](symbol[,symbol]...)|module}]...
```

- 省略時解釈
  なし

[詳細説明]

- 外部シンボル名 *symbol*，またはライブラリ・モジュール *module* を削除します。
- 特定のファイル *file* 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 変数名，C 関数名を指定する場合は，プログラム中での定義名の先頭に “_” を付加します。
- 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の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。

[使用例]

- 全ファイル中のシンボル名 _sym1 を削除します。

```
>rlink a.obj -delete=(_sym1)
```

- b.obj 内のシンボル名 _sym2 を削除します。

```
>rlink a.obj b.obj -delete=b.obj(_sym2)
```

[備考]

- 本オプションは，-extract オプション，または -strip オプションと同時に指定した場合は無効となります。
- -form=library オプションを指定した場合は，ライブラリ・モジュールを削除することができます。
- -form={absolute|relocate|hexadecimal|stype|binary} オプションを指定した場合は，外部シンボルを削除することができます。
- コンパイル・オプション -Xmerge_files と本オプションを組み合わせて使用した場合，動作は保証されません。

-REPlace

ライブラリ・モジュールを置換します。

[指定形式]

```
-REPlace=file[(module[,module]...)][,file[(module[,module]...)]]...
```

- 省略時解釈
  なし

[詳細説明]

- 指定したファイル *file*，またはライブラリ・モジュール *module* と -library オプションで指定したライブラリ・ファイル内の同名モジュールを置換します。

[使用例]

- file1.obj とライブラリ・ファイル lib1.lib 内のモジュール file1 を置換します。

```
>rlink -library=lib1.lib -replace=file1.obj -form=library
```

- モジュール mdl1 とライブラリ・ファイル lib1.lib 内のモジュール mdl1 を置換します。

```
>rlink -library=lib1.lib,lib2.lib -replace=lib1.lib(mdl1) -form=library
```

[備考]

- 本オプションは，-form={object|relocate|absolute|hexadecimal|stype|binary} オプション，および -extract，-strip オプションを指定した場合は無効となります。
- コンパイル・オプション -Xmerge_files と本オプションを組み合わせて使用した場合，動作は保証されません。

-EXTract

ライブラリ・モジュールを抽出します。

[指定形式]

```
-EXTract=module[,module]...
```

- 省略時解釈
  なし

[詳細説明]

- ライブラリ・モジュール *module* 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。

[使用例]

- ライブラリ・ファイル lib1.lib からモジュール file1 を抽出し，オブジェクト・ファイル出力形式のファイルを出力します。

```
>rlink -library=lib1.lib -extract=file1 -form=obj
```

[備考]

- 本オプションは，-form={absolute|hexadecimal|stype|binary} オプション，および -strip オプションを指定した場合は無効となります。
- -form=library オプションを指定した場合は，ライブラリ・モジュールを削除することができます。
- -form={absolute|relocate|hexadecimal|stype|binary} オプションを指定した場合は，外部シンボルを削除することができます。

-STRip

ロード・モジュール・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。

[指定形式]

```
-STRip
```

- 省略時解釈
  なし

[詳細説明]

- ロード・モジュール・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。
- デバッグ情報を削除する前のファイルは，“ *ファイル名* .abk ” という名前のファイルにバックアップします。
- 本オプションを指定した場合，入力ファイルと出力ファイルは 1 対 1 対応になります。

[使用例]

- 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。デバッグ情報を削除する前のファイルは，file1.abk，file2.abk，file3.abk にバックアップします。
  ＜コマンド・ライン＞

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

＜サブコマンド・ファイル sub.txt ＞

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

[備考]

- 本オプションは，-form={object|relocate|hexadecimal|stype|binary} オプションを指定した場合は無効となります。

-CHange_message

インフォメーション，ワーニング，エラーのメッセージ種別を変更します。

[指定形式]

```
-CHange_message=level[=num[-num][,num[-num]]...][,level[=num[-num][,num[-num]]...]]
...
```

- 省略時解釈
  なし

[詳細説明]

- インフォメーション，ワーニング，エラーのメッセージ種別 *level* を変更します。
- メッセージ出力時の処理継続／中断を変更することができます。
- *level* に指定可能なものを以下に示します。

Information	インフォメーション
Warning	ワーニング
Error	エラー

- メッセージ番号 *num* 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。
- *num* 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。

[使用例]

- E0561310 をワーニングに変更し，E0561310 出力時も処理を継続します。

```
>rlink a.obj b.obj -change_message=warning=1310
```

- すべてのインフォメーション，ワーニングをエラーに変更します。
  メッセージを 1 つでも出力すると，処理を中断します。

```
>rlink a.obj b.obj -change_message=error
```

-Hide

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。

[指定形式]

```
-Hide
```

- 省略時解釈
  なし

[詳細説明]

- 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るため，バイナリ・エディタなどでファイルを開いてもローカル・シンボル名を確認することができなくなります。
  なお，生成されるファイルの動作に対する影響は一切ありません。
- 本オプションは，ローカル・シンボル名を機密扱いにしたい場合などに指定してください。
- 秘匿対象となるシンボルの種類を以下に示します。
  - C ソース：static 型修飾子を指定した変数名，関数名など
  - C ソース：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
    なお，エントリ関数名は秘匿対象にはなりません。

[使用例]

- 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。

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

本オプションの機能が有効となる C ソース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，-form={absolute|relocate|library} オプションを指定した場合のみ有効となります。
- 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

リンク後のセクションの合計サイズを標準エラー出力に表示します。

[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エラー出力に表示します。
- 以下の 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
- 本オプションを指定することにより，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。

[使用例]

- リンク後のセクションの合計サイズを標準エラー出力に表示します。

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

[備考]

- リンク・マップ・ファイルに合計サイズを表示するためには，-show=total_size オプションを指定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。

-LOgo

コピーライトを出力します。

[指定形式]

```
-LOgo
```

- 省略時解釈
  コピーライトを出力します。

[詳細説明]

- コピーライトを出力します。

[使用例]

- コピーライトを出力します。

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

-NOLOgo

コピーライトの出力を抑止します。

[指定形式]

```
-NOLOgo
```

- 省略時解釈
  コピーライトを出力します（-logo オプションの指定と同じです）。

[詳細説明]

- コピーライトの出力を抑止します。

[使用例]

- コピーライトの出力を抑止します。

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

-END

本オプションより前に指定したオプション列を実行します。

[指定形式]

```
-END
```

- 省略時解釈
  なし

[詳細説明]

- 本オプションより前に指定したオプション列を実行します。
  リンク処理の終了後に，本オプションより後に指定したオプション列の入力，リンク処理を継続します。

[注意]

- 本オプションは，サブコマンド・ファイル内のみで使用することができます。

[使用例]

- サブコマンド・ファイル test.sub を以下の内容で作成します。

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

コマンド・ライン上でサブコマンド・ファイル test.sub を指定します。

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

（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

オプション指定の終了を指定します。

[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。

[注意]

- 本オプションは，サブコマンド・ファイル内のみで使用することができます。

[使用例]

- サブコマンド・ファイル test.sub を以下の内容で作成します。

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

コマンド・ライン上でサブコマンド・ファイル test.sub を指定します。

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

（1）～（3）の処理を実行し，a.abs を出力します。
本オプションの実行後にコマンド・ライン上で指定している -nodebug オプションは無効になります。

2.6 オプションの複数指定

ここでは，ccrh コマンドに対してオプションを同時に 2 つ以上指定した場合について説明します。

2.6.1 優先順序

以下のオプションは，ほかの特定のオプションを無効とします。

-V/-h	ほかのすべてのオプションは無効となります。 このとき，警告を出力しません。
-P	プリプロセス処理で終了するため，それ以降の処理に関するオプションは無効となります。 このとき，警告を出力します。
-S	コンパイル処理で終了するため，それ以降の処理に関するオプションは無効となります。 このとき，警告を出力します。
-c	アセンブル処理で終了するため，それ以降の処理に関するオプションは無効となります。 このとき，警告を出力します。
-Xcpu=g3k	-Xfloat オプションは無効となります。 このとき，警告を出力します。

以下の組み合わせでオプションを指定した場合は，警告を出力して，最後に指定したものが有効となります。

- -P，-S，-c
- -D，-U（シンボル名が同じ場合）
- -Onothing，-Odefault，-Osize，-Ospeed

なお，オプションの指定順序により，以下のオプションは無効となります。

- -Onothing，-Odefault，-Osize，-Ospeed の前に指定した -O*item* 注

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

2.6.2 機能矛盾

以下の組み合わせでオプションを指定した場合は，エラーとなります。

- -Omap，-Xep=fix
- -Osmap，-Xep=fix
- -Omap と -Xsection=data=ep_disp16 もしくは ep_disp23
- -Osmap と -Xsection=data=ep_disp16 もしくは ep_disp23

2.6.3 依存関係

以下のオプションは，ほかの特定のオプションにより動作が変わります。

-Xpreprocess	-P オプションと同時に指定しない場合，無効となります。 このとき，警告を出力しません。
-o	-P/-S/-c オプションと同時に指定した場合，生成ファイルの種別は，それぞれプリプロセス処理済みファイル／アセンブリ・ソース・ファイル／オブジェクト・ファイルとなります。
-g	-O オプションと同時に指定した場合，デバッグ情報が正しくないことがあります。
-Oinline	-Xmerge_files オプションと同時に指定した場合，ファイル間インライン展開を行う場合があります。

2.6.4 #pragma 指定との関係

以下のオプションは，#pragma 指定との関係により動作が変わります。

- -Xep=callee，または -Xep の指定なし
  #pragma section で以下の属性指定文字を同時に指定した場合は，エラーとなります。
  edata，edata23，tdata，tdata4，tdata5，tdata7，tdata8，sdata，sdata23，ep_auto，ep_disp4，ep_disp5，ep_disp7，ep_disp8，ep_disp16，ep_disp23，gp_disp16，gp_disp23

3. 出力ファイル

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。

3.1 アセンブル・リスト・ファイル

ここでは，アセンブル・リスト・ファイルについて説明します。
アセンブル・リストとは，ソースをコンパイル，アセンブルして出力するコードをリスト形式にしたものです。
これにより，コンパイル，アセンブルした結果が，どのようなコードになっているかを確認することができます。

3.1.1 アセンブル・リストの構成

アセンブル・リストの構成と内容を以下に示します。

出力情報	説明
アセンブル・リスト	アセンブル時のロケーション・カウンタ値，コード，行番号，ソース・プログラム
セクション・リスト	セクションの種類，サイズ，名称
コマンド・ライン情報	アセンブラのコマンド・ライン文字列

3.1.2 アセンブル・リスト

アセンブル時のロケーション・カウンタ値，コード，行番号，ソース・プログラムを出力します。
アセンブル・リストの出力例を以下に示します。

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

00000000                       1  #CC-RH Compiler RH850 Assembler Source
00000000                       2  #@      CC-RH Version : VX.XX.XXx [XX Xxx XXXX]
00000000                       3  #@      Command : main.c -Xcommon=v850e3v5 -S
00000000                       4  #@      compiled at Xxx Xxx XX XX:XX:XX XXXX
00000000                       5          .cseg text
00000000                       6          ld.w $_data,r12
00000000  440E0000              -- movhi       0x0,gp,r1
00000004  21670100              -- ld.w        0x0[r1],r12
00000008                       7
00000000                       8          .dseg data
00000000  00000000             9  _data:  .dw 0
00000004                      10
```

項番	説明
(1)	ロケーション・カウンタ値 その行のソース・プログラムに対して生成したコードの先頭に対するロケーション・カウンタ値を出力します。
(2)	コード その行のソース・プログラムに対して生成したコード（機械語命令，またはデータ）を出力します。 1 バイトごとに 2 桁の 16 進数で表記します。
(3)	行番号 その行の行番号を出力します。 10 進数で表記します。

項番	説明
(4)	ソース・プログラム その行のソース・プログラムを出力します。 その行の命令に対して命令展開が生じた場合，その命令展開によって生成した機械語命令の命令列を逆アセンブルした結果を，“--” 以降に示します。 コンパイラ情報（1 ～ 4 行目）は，コンパイラが出力したアセンブリ・ソース・ファイルをアセンブルした場合のみ出力します。

3.1.3 セクション・リスト

セクションの種類，サイズ，名称を出力します。
セクション・リストの出力例を以下に示します。

```
Section List
(1)     (2)                 (3)
Attr    Size                Name

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

項番	説明
(1)	セクションの種類 セクションの種類を再配置属性として出力します。
(2)	セクション・サイズ セクションのサイズを出力します（単位：バイト）。 10 進数，およびかっこ内に 16 進数で表記します。
(3)	セクション名 セクション名を出力します。

3.1.4 コマンド・ライン情報

アセンブラのコマンド・ライン文字列を出力します。
コマンド・ライン情報の出力例を以下に示します。

```
Command Line Parameter

a.asm -Xcommon=v850e3v5 -Xprn_path      (1)
```

項番	説明
(1)	コマンド・ライン文字列 アセンブラに指定したコマンド・ライン文字列を出力します。

3.2 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。

3.2.1 リンク・マップの構成

リンク・マップの構成と内容を以下に示します。

出力情報	説明	-show オプション指定	-show オプション省略時
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	―	出力する
エラー情報	エラー・メッセージ	―	出力する
リンク・マップ情報	セクション名，先頭／最終アドレス，サイズ，種別	―	出力する
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	-show=total_size	出力しない
シンボル情報	静的定義シンボル名，アドレス，サイズ，種別（アドレス順），最適化実行の有無 -show=reference を指定した場合は，各シンボルの参照回数も出力します。 show=struct を指定した場合は，構造体／共用体メンバのアドレスも出力します。	-show=symbol -show=reference	出力しない
クロス・リファレンス情報	シンボルの参照情報	-show=xreference	出力しない

注意 -show オプションは，-list オプションを指定した場合に有効となります。
-show オプションについての詳細は，「-SHow」を参照してください。

3.2.2 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。
コマンド・ライン，サブコマンド・ファイルで次のように指定した場合のオプション情報の出力例を以下に示します。

＜コマンド・ライン＞

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

＜サブコマンド・ファイル test.sub ＞

```
input sample.obj
```

```
*** Options ***


-subcommand=test.sub     (1)

input sample.obj         (2)

-list                    (1)

-show                    (1)
```

項番	説明
(1)	コマンド・ラインで指定したオプション コマンド・ラインで指定したオプションを出力します（指定順）。
(2)	サブコマンド・ファイル内で指定したオプション サブコマンド・ファイル test.sub 内で指定したオプションを出力します。

3.2.3 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以下に示します。

```
*** Error information ***

** E0562310 Undefined external symbol "_func_02" referenced in "sample.obj"     (1)
```

項番	説明
(1)	エラー・メッセージ エラー・メッセージを出力します。

3.2.4 リンク・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。
リンク・マップ情報の出力例を以下に示します。

```
*** Mapping List ***

(1)                                (2)        (3)        (4)     (5)
SECTION                            START      END        SIZE    ALIGN

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

項番	説明
(1)	セクション名 セクション名を出力します。
(2)	先頭アドレス 先頭アドレスを出力します。 16 進数で表記します。

項番	説明
(3)	最終アドレス 最終アドレスを出力します。 16 進数で表記します。
(4)	セクション・サイズ セクション・サイズを出力します（単位：バイト）。 16 進数で表記します。
(5)	セクションのアライメント数 セクションのアライメント数を出力します。

3.2.5 合計セクション・サイズ

-show=total_size オプションを指定した場合，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。
合計セクション・サイズの出力例を以下に示します。

```
*** 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 RAM データ・セクションの合計サイズを出力します。 16 進数で表記します。
(2)	ROM データ・セクションの合計サイズ ROM データ・セクションの合計サイズを出力します。 16 進数で表記します。
(3)	プログラム・セクションの合計サイズ プログラム・セクションの合計サイズを出力します。 16 進数で表記します。

3.2.6 シンボル情報

-show=symbol オプションを指定した場合は，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，最適化実行の有無をアドレス順で出力します。
また，-show=reference オプションを指定した場合は，各シンボルの参照回数も出力します。
シンボル情報の出力例を以下に示します。

```
*** Symbol List ***

SECTION=(1)
FILE=(2)
                (3)         (4)             (5)
                START       END             SIZE
  (6)           (7)         (8)             (9)          (10)    (11)
  SYMBOL        ADDR        SIZE            INFO         COUNTS  OPT

SECTION=.text
FILE=sample.obj
                00000000    00000023        24
  _main
                00000000          0         func ,g            0
  _func_01
                00000018          0         func ,g            0
SECTION=.bss
FILE=sample.obj
                fe600004    fe60000b        8
  _gvall
                fe600004          4         data ,g            0
```

項番	説明
(1)	セクション名 セクション名を出力します。
(2)	ファイル名 ファイル名を出力します。

項番	説明
(3)	先頭アドレス (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出力します。 16 進数で表記します。
(4)	最終アドレス (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出力します。 16 進数で表記します。
(5)	セクション・サイズ (2) のファイルに含まれる該当セクションのセクション・サイズを出力します（単位：バイト）。 16 進数で表記します。
(6)	シンボル名 シンボル名を出力します。
(7)	シンボル・アドレス シンボル・アドレスを出力します。 16 進数で表記します。
(8)	シンボル・サイズ シンボル・サイズを出力します（単位：バイト）。 16 進数で表記します。
(9)	シンボル種別 データ種別，および宣言種別を出力します。 - 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 シンボル参照回数を出力します。 16 進数で表記します。 -show=reference オプションを指定した場合のみ出力します。 参照回数を出力しないときは，“*” を出力します。
(11)	最適化実行の有無 最適化実行の有無を出力します。 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

show=struct を指定した場合は，コンパイル時に -g を指定したファイル内で定義した構造体／共用体メンバの情報も出力します。構造体メンバ情報の出力例を以下に示します。

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

項番	説明
(12)	構造体の場合は struct，共用体の場合は union
(13)	構造体または共用体全体のサイズ
(14)	メンバ名をシンボル名に “.”（ドット）で連結して表示
(15)	メンバアドレスを表示
(16)	メンバのサイズを表示
(17)	メンバの型を表示

3.2.7 クロス・リファレンス情報

-show=xreference オプションを指定した場合，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以下に示します。

```
*** Cross Reference List ***

(1)  (2)        (3)            (4)       (5)
No   Unit Name  Global.Symbol  Location  External Information
0001 sample1
     SECTION=.text
                _main
                               00000000
                _func_01
                               00000018
     SECTION=.data
                _gval3
                               fe600000  0003(00000032:.text)
                                         0003(00000038:.text)
     SECTION=.bss
                _gval1
                               fe600004  0001(0000001a:.text)
                                         0001(00000020:.text)
                _gval2
                               fe600008  0002(00000026:.text)
                                         0002(0000002c:.text)
0002 sample2
     SECTION=.text
                _func02
                               00000024  0001(0000000a:.text)
0003 sample3
     SECTION=.text
                _func03
                               00000030  0001(00000010:.text)
```

項番	説明
(1)	Unit 番号 オブジェクト単位の識別番号を出力します。
(2)	オブジェクト名 オブジェクト名をリンク時の入力指定順で出力します。
(3)	シンボル名 シンボル名をセクションごとに配置アドレスの昇順で出力します。
(4)	シンボルの配置アドレス シンボルの配置アドレスを出力します。 -form=relocate オプションを指定した場合は，セクションの先頭からの相対値となります。
(5)	参照している外部シンボルのアドレス 参照している外部シンボルのアドレスを以下の形式で出力します。 *Unit 番号 (アドレス，または セクション内オフセット : セクション名)*

3.3 リンク・マップ・ファイル（オブジェクト結合時）

ここでは，マルチコア用プロジェクトにおいて，オブジェクト結合機能を使用する場合に出力するリンク・マップ・ファイルについて説明します。
リンク・マップ・ファイルを出力することにより，オブジェクト結合の詳細情報を知ることができます。

3.3.1 リンク・マップの構成

リンク・マップの構成と内容を以下に示します。

出力情報	説明
ヘッダ情報	最適化リンカのバージョン情報，およびリンク時刻
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
エラー情報	エラー・メッセージ
エントリ情報	実行開始アドレス
結合アドレス情報	結合元ファイル，および連続範囲データの開始／終了アドレス，サイズ
アドレス重複情報	重複した結合元ファイル，および重複範囲データの開始／終了アドレス，サイズ

3.3.2 ヘッダ情報

バージョン情報，およびリンク時刻を出力します。
ヘッダ情報の出力例を以下に示します。

```
Renesas Optimizing Linker (VX.XX.XX)            XX-Xxx-XXXX XX:XX:XX     (1)
```

項番	説明
(1)	最適化リンカのバージョン情報，およびリンク時刻 最適化リンカのバージョン情報，およびリンク時刻を出力します。

3.3.3 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。
コマンド・ライン，サブコマンド・ファイルで次のように指定した場合のオプション情報の出力例を以下に示します。

＜コマンド・ライン＞

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

＜サブコマンド・ファイル test.sub ＞

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

```
*** Options ***

-subcommand=test.sub     (1)
input sample1.mot        (2)
input sample2.mot        (2)
form stype               (2)
output result            (2)
-list                    (1)
```

項番	説明
(1)	コマンド・ラインで指定したオプション コマンド・ラインで指定したオプションを出力します（指定順）。
(2)	サブコマンド・ファイル内で指定したオプション サブコマンド・ファイル test.sub 内で指定したオプションを出力します。

3.3.4 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以下に示します。

```
*** Error information ***
E0562420:"sample1.mot" overlap address "sample2.mot" : "00000100"       (1)
```

項番	説明
(1)	エラー・メッセージ エラー・メッセージを出力します。

3.3.5 エントリ情報

実行開始アドレスを出力します。
エントリ情報の出力例を以下に示します。

```
*** Entry address ***
00000100        (1)
```

項番	説明
(1)	実行開始アドレス 実行開始アドレスを出力します。 ただし，実行開始アドレスが “00000000” の場合は出力しません。

3.3.6 結合アドレス情報

結合元ファイル，および連続範囲データの開始／終了アドレス，サイズを出力します。
結合アドレス情報の出力例を以下に示します。

```
*** Combine information ***
(1)             (2)         (3)         (4)
FILE            START       END         SIZE
sample1.mot
                00000100    00000127    28
sample1.mot
                00000200    00000227    28
sample2.mot
                00000250    00000263    14
sample2.mot
                00000300    0000033b    3c
```

項番	説明
(1)	結合元ファイル名 結合元ファイル名を出力します。
(2)	連続範囲データの開始アドレス 連続範囲データの開始アドレスを出力します。 16 進数で表記します。
(3)	連続範囲データの終了アドレス 連続範囲データの終了アドレスを出力します。 16 進数で表記します。
(4)	連続範囲データのサイズ 連続範囲データのサイズを出力します（単位：バイト）。 16 進数で表記します。

3.3.7 アドレス重複情報

重複した結合元ファイル，および重複範囲データの開始／終了アドレス，サイズを出力します。
アドレス重複情報の出力例を以下に示します。

```
*** Conflict information ***
(1)             (2)         (3)         (4)
FILE            START       END         SIZE
Conflict 1
                00000200    00000213    14
sample1.mot
sample2.mot
```

項番	説明
(1)	重複した結合元ファイル名 重複した結合元ファイル名を出力します。
(2)	重複範囲データの開始アドレス 重複範囲データの開始アドレスを出力します。 16 進数で表記します。
(3)	重複範囲データの終了アドレス 重複範囲データの終了アドレスを出力します。 16 進数で表記します。
(4)	重複範囲データのサイズ 重複範囲データのサイズを出力します（単位：バイト）。 16 進数で表記します。

3.4 ライブラリ・リスト・ファイル

ここでは，ライブラリ・リスト・ファイルについて説明します。
ライブラリ・リストとは，ライブラリ作成結果の情報が書かれたものです。

3.4.1 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以下に示します。

出力情報	説明	-show オプション指定	-show オプション省略時
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	－	出力する
エラー情報	エラー・メッセージ	－	出力する
ライブラリ情報	ライブラリ情報	－	出力する
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	－	出力する
	モジュール内シンボル名	-show=symbol	出力しない
	各モジュール内セクション名，シンボル名	-show=section	出力しない

注意 -show オプションは，-list オプションを指定した場合に有効となります。
-show オプションについての詳細は，「-SHow」を参照してください。

3.4.2 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。
コマンド・ライン，サブコマンド・ファイルで次のように指定した場合のオプション情報の出力例を以下に示します。

＜コマンド・ライン＞

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

＜サブコマンド・ファイル test.sub ＞

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

```
*** Options ***

-subcommand=test.sub     (1)
form library             (2)
input extmod1            (2)
input extmod2            (2)
output usrlib.lib        (2)
-list                    (1)
-show                    (1)
```

項番	説明
(1)	コマンド・ラインで指定したオプション コマンド・ラインで指定したオプションを出力します（指定順）。

項番	説明
(2)	サブコマンド・ファイル内で指定したオプション サブコマンド・ファイル test.sub 内で指定したオプションを出力します。

3.4.3 エラー情報

エラー，ワ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以下に示します。

```
*** Error Information ***

** E0561200 Backed up file "sample1.lib" into "usrlib.lbk"      (1)
```

項番	説明
(1)	メッセージ メッセージを出力します。

3.4.4 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以下に示します。

```
*** Library Information ***

LIBRARY NAME=usrlib.lib (1)
CPU=RH850               (2)
ENDIAN=Little           (3)
ATTRIBUTE=user          (4)
NUMBER OF MODULE=2      (5)
```

項番	説明
(1)	ライブラリ名 ライブラリ名を出力します。
(2)	マイコン名 マイコン名を出力します。
(3)	エンディアン種別 エンディアン種別を出力します。
(4)	ライブラリ・ファイルの属性 システム・ライブラリであるか，ユーザ・ライブラリであるかを出力します。
(5)	ライブラリ内モジュール数 ライブラリ内モジュール数を出力します。

3.4.5 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を出力します。
-show=symbol オプションを指定した場合は，モジュール内シンボル名を出力します。
また，-show=section オプションを指定した場合は，モジュール内セクション名も出力します。
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以下に示します。

```
*** Library List ***

(1)             (2)
MODULE          LAST UPDATE
  (3)
  SECTION
    (4)
    SYMBOL
extmod1
                12-Dec-2011 16:30:00
  .text
    _func_01
    _func_02
extmod2
                12-Dec-2011 16:30:10
  .text
    _func_03
    _func_04
```

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

3.5 インテル拡張ヘキサ・ファイル

ここでは，インテル拡張ヘキサ・ファイルについて説明します。

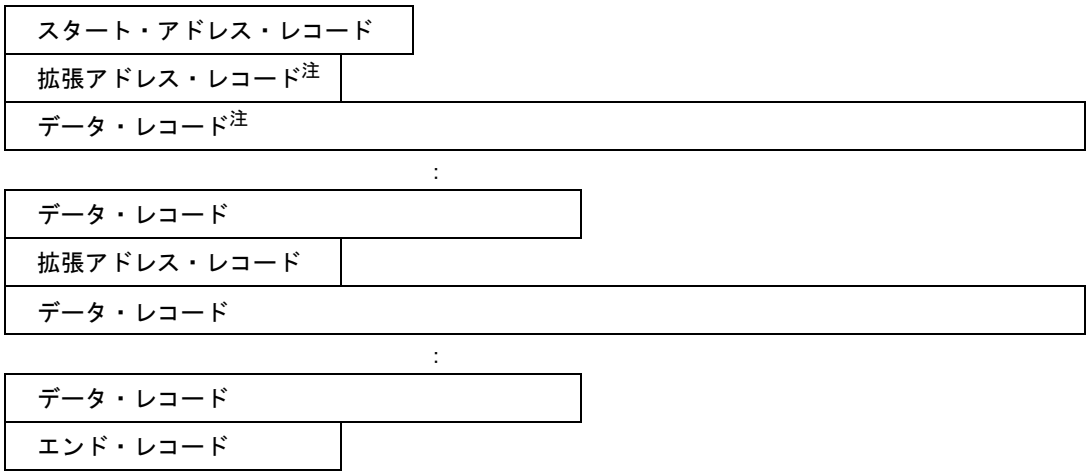
3.5.1 インテル拡張ヘキサ・ファイルの構成

インテル拡張ヘキサ・ファイル（20 ビット）は，スタート・アドレス・レコード，拡張アドレス・レコード，データ・レコード，およびエンド・レコードの 4 種類のレコード注により構成されます。
インテル拡張ヘキサ・ファイル（32 ビット）は，スタート・リニア・アドレス・レコード，拡張リニア・スタート・アドレス・レコード，スタート・アドレス・レコード，拡張アドレス・レコード，データ・レコード，およびエンド・レコードの 6 種類のレコード注により構成されます。

注 各レコードは，ASCII コードで出力します。

インテル拡張ヘキサ・ファイルの構成と内容を以下に示します。

図 3.1 インテル拡張ヘキサ・ファイルの構成



注 拡張アドレス・レコード，およびデータ・レコードは繰り返されます。

出力情報	説明
スタート・リニア・アドレス・レコード	リニア・アドレス
拡張リニア・アドレス・レコード	ビット 32 ～ビット 16 の上位 16 ビット・アドレス
スタート・アドレス・レコード	エントリ・ポイント・アドレス
拡張アドレス・レコード	ロード・アドレスのパラグラフ値
データ・レコード	コードの値
エンド・レコード	コードの終わり

各レコードは，各種フィールドにより以下の形で構成されます。

```
:   XX  XXXX  XX  DD......DD SS  NL
(1) (2) (3)   (4)     (5)    (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	バイト数 (5) の 2 桁ずつの 16 進数で表されるバイトのバイト数です。

項番	説明
(3)	ロケーション・アドレス
(4)	レコード・タイプ 05 : スタート・リニア・アドレス・レコード 04 : 拡張リニア・アドレス・レコード 03 : スタート・アドレス・レコード 02 : 拡張アドレス・レコード 00 : データ・レコード 01 : エンド・レコード
(5)	コード コードの 1 バイトごとを 2 桁の 16 進数で表したものです。
(6)	チェック・サム ：，SS，NL を除くレコード内の各バイト値を 16 進数で加算した結果の 2 の補数です（2 桁）。
(7)	ニュー・ライン（¥n）

備考 インテル・ヘキサ・フォーマットのロケーション・アドレスは 2 バイト（16 ビット）です。
したがって，64K の空間しか直接指定はできません。
それを拡張するために，16 ビットの拡張アドレスを追加して 1M（20 ビット）の空間まで扱えるようにしたのがインテル拡張ヘキサ・フォーマットです。
具体的には，16 ビットの拡張アドレスを指定するレコード・タイプを追加しています。
この追加した拡張アドレスの 4 ビットをシフトしてロケーション・アドレスと加算することで，20 ビットのアドレスを表現できるようになっています。

3.5.2 スタート・リニア・アドレス・レコード

リニア・アドレスを示します。

```
:   04  0000  05  XXXXXXXX  SS  NL
(1) (2) (3)   (4)    (5)    (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	04 固定
(3)	0000 固定
(4)	レコード・タイプ（05 固定）
(5)	リニア・アドレス値
(6)	チェック・サム
(7)	ニュー・ライン

3.5.3 拡張リニア・アドレス・レコード

ビット 32 ～ビット 16 の上位 16 ビット・アドレスを示します。

```
:   02  0000  04  XXXX  SS  NL
(1) (2) (3)   (4)  (5)  (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	02 固定

項番	説明
(3)	0000 固定
(4)	レコード・タイプ（04 固定）
(5)	ビット 32 ～ビット 16 の上位 16 ビット・アドレス値
(6)	チェック・サム
(7)	ニュー・ライン

注　下位 16 ビットは，データ・レコードのロケーション・アドレスを用います。

3.5.4 スタート・アドレス・レコード

エントリ・ポイント・アドレスを示します。

```
:   04  0000  03  PPPP  XXXX  SS  NL
(1) (2) (3)   (4)  (5)   (6)  (7) (8)
```

項番	説明
(1)	レコード・マーク
(2)	04 固定
(3)	0000 固定
(4)	レコード・タイプ（03 固定）
(5)	エントリ・ポイント・アドレスのパラグラフ値[注]
(6)	エントリ・ポイント・アドレスのオフセット値
(7)	チェック・サム
(8)	ニュー・ライン

注　アドレスは（パラグラフ値 <<4）＋オフセット値で求められます。

3.5.5 拡張アドレス・レコード

ロード・アドレスのパラグラフ値を示します[注]。

注　（データ・レコードを出力する際）セグメントの先頭で，またはデータ・レコードのロード・アドレスのオフセット値が最大値 0xffff を越えてセグメントが新しくなる際，出力します。

```
:   02  0000  02  PPPP  SS  NL
(1) (2) (3)   (4)  (5)  (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	02 固定
(3)	0000 固定
(4)	レコード・タイプ（02 固定）
(5)	セグメントのパラグラフ値
(6)	チェック・サム

項番	説明
(7)	ニュー・ライン

3.5.6 データ・レコード

コードの値を示します。

```
:   XX  XXXX  00  DD......DD  SS  NL
(1) (2) (3)   (4)     (5)     (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	バイト数注
(3)	ロケーション・アドレス
(4)	レコード・タイプ（00 固定）
(5)	コード コードの 1 バイトごとを 2 桁の 16 進数で表したものです。
(6)	チェック・サム
(7)	ニュー・ライン

注 0x1 ～ 0xff の範囲に限られます（1 つのデータ・レコードで示されるコードのバイト数の最小値は 1 で最大値は 255 です）。

例

```
:   04  0100  00  3C58E01B  6C  NL
(1) (2) (3)   (4)    (5)    (6) (7)
```

項番	説明
(1)	レコード・マーク
(2)	3C58E01B の 2 桁ずつの 16 進数で表されるバイトのバイト数
(3)	ロケーション・アドレス
(4)	レコード・タイプ 00
(5)	コードの 1 バイトごとを 2 桁の 16 進数で表したもの
(6)	チェック・サム 04 ＋ 01 ＋ 00 ＋ 00 ＋ 3C ＋ 58 ＋ E0 ＋ 1B ＝ 194 の 2 の補数 E6C の下位 1 バイトを 2 桁の 16 進数で表したもの
(7)	ニュー・ライン（¥n）

3.5.7 エンド・レコード

コードの終わりを示します。

```
:   00  0000  01  FF  NL
(1) (2) (3)   (4) (5) (6)
```

項番	説明
(1)	レコード・マーク
(2)	00 固定
(3)	0000 固定
(4)	レコード・タイプ（01 固定）
(5)	FF 固定
(6)	ニュー・ライン

3.6 モトローラ・S タイプ・ファイル

ここでは，モトローラ・S タイプ・ファイルについて説明します。

3.6.1 モトローラ・S タイプ・ファイルの構成

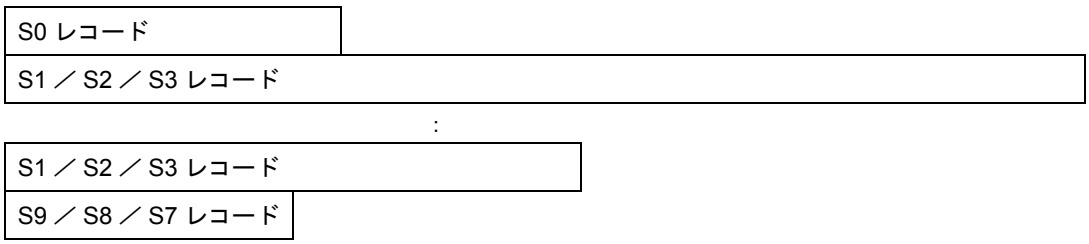
モトローラ・S タイプ・ファイルは，ヘッダ・レコードである S0 レコード，データ・レコードである S1 ／ S2 ／ S3 レコード，エンド・レコードである S9 ／ S8 ／ S7 レコードの 7 種類のレコード[注 1] により構成されます[注 2]。

注 1. 各レコードは，ASCII コードで出力します。

注 2. モトローラ・S タイプ・ファイルには，16 ビット・アドレスのもの，（24 ビット）スタンダード・アドレスのもの，32 ビット・アドレスのものが存在し，16 ビット・アドレスのフォーマットは S0，S1，および S9 レコード，スタンダード・アドレスのフォーマットは S0，S2，および S8 レコード，32 ビット・アドレスのフォーマットは S0，S3，および S7 レコードによって構成されます。

モトローラ・S タイプ・ファイルの構成と内容を以下に示します。

図 3.2 モトローラ・S タイプ・ファイルの構成



出力情報	説明
S0 レコード	ファイル名
S1 レコード	コードの値
S2 レコード	コードの値
S3 レコード	コードの値
S7 レコード	エントリ・ポイント・アドレス
S8 レコード	エントリ・ポイント・アドレス
S9 レコード	エントリ・ポイント・アドレス

各レコードは，各種フィールドにより以下の形で構成されます。

```
Sx  XX  YY......YY  SS  NL
(1) (2)     (3)     (4) (5)
```

項番	説明
(1)	レコード・タイプ S0 : S0 レコード S1 : S1 レコード S2 : S2 レコード S3 : S3 レコード S4 : S4 レコード S5 : S5 レコード S6 : S6 レコード S7 : S7 レコード S8 : S8 レコード S9 : S9 レコード

項番	説明
(2)	レコード長 (3) の 2 桁ずつの 16 進数で表されるバイトのバイト数＋ SS で表されるバイト数[注]
(3)	フィールド
(4)	チェック・サム Sx，SS，NL を除くレコード内の 2 桁ずつの 16 進数で表されるバイトの値を合計したものの 1 の補数を取り，その下位 1 バイトを 2 桁の 16 進数で表したもの
(5)	ニュー・ライン（¥n）

注 1 です。

3.6.2 S0 レコード

ファイル名を示します。

```
S0  0E  0000  XX......XX  SS  NL
(1) (2) (3)      (4)     (5) (6)
```

項番	説明
(1)	S0 固定
(2)	0E 固定
(3)	0000 固定
(4)	主にファイル名（8 文字）＋ファイル形式（3 文字）
(5)	チェック・サム
(6)	ニュー・ライン

3.6.3 S1 レコード

コードの値を示します。

```
S1  XX  YYYY  ZZ......ZZ  SS  NL
(1) (2) (3)      (4)     (5) (6)
```

項番	説明
(1)	S1 固定
(2)	レコード長
(3)	ロード・アドレス 16 ビット（0x0 ～ 0xFFFF）
(4)	コード コードの 1 バイトごとを 2 桁の 16 進数で表したもの
(5)	チェック・サム
(6)	ニュー・ライン

3.6.4 S2 レコード

コードの値を示します。

```
S2  XX  YYYYYY  ZZ......ZZ  SS  NL
(1) (2)   (3)       (4)     (5) (6)
```

項番	説明
(1)	S2 固定
(2)	レコード長
(3)	ロード・アドレス 24 ビット（0x0 ～ 0xFFFFFF）
(4)	コード コードの 1 バイトごとを 2 桁の 16 進数で表したもの
(5)	チェック・サム
(6)	ニュー・ライン

3.6.5 S3 レコード

コードの値を示します。

```
S3  XX  YYYYYYYY  ZZ......ZZ  SS  NL
(1) (2)   (3)         (4)     (5) (6)
```

項番	説明
(1)	S3 固定
(2)	レコード長
(3)	ロード・アドレス 32 ビット（0x0 ～ 0xFFFFFFFF）
(4)	コード コードの 1 バイトごとを 2 桁の 16 進数で表したもの
(5)	チェック・サム
(6)	ニュー・ライン

3.6.6 S7 レコード

エントリ・ポイント・アドレスを示します。

```
S7  XX  YYYYYYYY  SS  NL
(1) (2)   (3)     (4) (5)
```

項番	説明
(1)	S7 固定
(2)	レコード長
(3)	エントリ・ポイント・アドレス 32 ビット（0x0 ～ 0xFFFFFFFF）
(4)	チェック・サム
(5)	ニュー・ライン

3.6.7 S8 レコード

エントリ・ポイント・アドレスを示します。

```
S8  XX  YYYYYY  SS  NL
(1) (2)   (3)   (4) (5)
```

項番	説明
(1)	S8 固定
(2)	レコード長
(3)	エントリ・ポイント・アドレス 24 ビット（0x0 ～ 0xFFFFFF）
(4)	チェック・サム
(5)	ニュー・ライン

3.6.8 S9 レコード

エントリ・ポイント・アドレスを示します。

```
S9  XX  YYYY  SS  NL
(1) (2)  (3)  (4) (5)
```

項番	説明
(1)	S9 固定
(2)	レコード長
(3)	エントリ・ポイント・アドレス 16 ビット（0x0 ～ 0xFFFF）
(4)	チェック・サム
(5)	ニュー・ライン

4. コンパイラ言語仕様

この章では，CC-RH がサポートするコンパイラ言語仕様（基本言語仕様，拡張言語仕様など）について説明します。

4.1 基本言語仕様

CC-RH は，ANSI 規格で規定された言語仕様をサポートしていますが，その中には処理系依存として規定されている項目があります。ここでは，ANSI 規格に規定／定義が行われていない項目，および処理系に依存した項目の言語仕様について説明します。

また，ANSI 規格に厳密にあわせて処理を行うオプション（-Xansi）を指定した場合と指定しない場合の差分についても説明します。

なお，CC-RH で独自に追加されている拡張言語仕様については，「4.2 拡張言語仕様」を参照してください。

4.1.1 未規定の項目

この項では，ANSI 規格で未規定の項目について説明します。

(1) 実行環境－ 静的記憶域の初期化
静的データは，コンパイル時にデータ・セクションとして出力されます。

(2) 文字表示の意味－行の最終位置，後退（¥b），水平タブ（¥t），垂直タブ（¥v）
表示装置に依存となります。

(3) 型－浮動小数点
IEEE754 注準拠しています。

注 IEEE：Institute of Electrical and Electronics Engineers（電気電信学会）の略称です。
また，IEEE754 とは，浮動小数点演算を扱うシステムにおいて，扱うデータ形式や数値範囲などの仕様の統一化を図った標準です。

(4) 式－評価順序
未規定とします。

(5) 関数呼び出し－引数の評価順序
未規定とします。

(6) 構造体指定子，および共用体指定子
ビット・フィールドを保持する構造体オブジェクトの境界調整は，「(8) ビット・フィールド」を参照してください。

(7) 関数定義－仮引数の記憶域
未規定とします。

(8) # 演算子
未規定とします。

4.1.2 未定義の項目

この項では，ANSI 規格における未定義の項目について説明します。

(1) 文字集合
ソース・ファイルに文字集合として定義された文字以外がある場合，メッセージを出力します。

(2) 字句要素
文字 “ ' ”，または文字 “ " ” がその最後の分類（区切り子，および字句的に他の前処理字句の種類に一致しない単一の非空白類文字）に入る場合，メッセージを出力します。

(3) 識別子
識別子全文字を意味がある文字とするため，意味のない文字は存在しません。

(4) 識別子の結合
翻訳単位の中で同じ識別子が内部結合と外部結合の両方で現れた場合，メッセージを出力します。

(5) 適合型と合成型
同じオブジェクト，または関数を参照するすべての宣言は，適合しなければなりません。それ以外の場合，メッセージを出力します。

(6) 文字定数

特定の非図形文字は，¥に続く英小文字から構成する拡張表記¥a，¥b，¥f，¥n，¥r，¥t，および¥vによって表現できます。その他の拡張表記はもたず，¥に続く文字は，その文字自身とします。

(7) 文字列リテラル－結合
単純文字列リテラルとワイド文字列リテラル字句が隣り合うとき，結合してワイド文字列リテラルとします。

(8) 文字列リテラル－変更
文字列リテラルの変更はユーザ責任となります。

(9) ヘッダ名
文字，'，"，//，または /* が，区切り記号＜と＞の間の文字列中，または二つの区切り記号 " の文字列中に現れた場合は，そのままファイル名として扱います。¥文字はフォルダ区切りとして扱います。

(10) 浮動小数点型と汎整数型
浮動小数点型の値を汎整数型に型変換する場合，整数部の値が汎整数型で表現できなければ，未定義とします。

(11) 左辺値及び関数指示子
不完全型が左辺値となった場合は，未定義とします。

(12) 関数呼び出し－引数の個数
実引数の数が少ない場合，未定義とします。

(13) 関数呼び出し－拡張後の引数の型
関数原型を含まない形で関数を定義し，かつ拡張後の実引数の型が，拡張後の仮引数の型と一致しない場合，仮引数は不定値となります。

(14) 関数呼び出し－適合しない型
呼び出される関数を表す式によって指される型と適合しない型で関数が定義されている場合，関数の戻り値は不正な値となります。

(15) 関数呼び出し－適合しない型
関数原型を含む型で関数を定義し，かつ拡張後の実引数の型が仮引数の型と適合しない場合，または関数原型が省略記号で終わっている場合，仮引数の型として解釈されます。

(16) アドレス，および間接演算子
正しくない値がポインタに代入されている場合の，単項 * 演算子の動作は，ハードウエア設計，および正しくない値の内容により，不定な値を取るか，不正なアクセスとなります。

(17) キャスト演算子－関数ポインタのキャスト
型変換されたポインタが元の型以外の関数を呼び出すために使われた場合，未定義とします。

(18) キャスト演算子－汎整数型のキャスト
ポインタを汎整数型にキャストした場合で，領域の大きさが不十分な場合は，未定義とします。

(19) 乗除演算子
コンパイル中に 0 による除算剰余算を検出した場合，メッセージを出力します。

(20) 加減演算子－配列以外のポインタ
配列オブジェクトの要素を指すかのように動作するもの以外のポインタに対して，加算，または減算を行っている場合，未定義とします。

(21) 加減演算子－別な配列へのポインタ減算
同じ配列オブジェクトの中を指すかのように動作するもの以外の 2 つのポインタに対し，減算を行っている場合，未定義とします。

(22) ビット単位のシフト演算子
右オペランドの値が負であるか，または拡張した左オペランドのビット幅以上の場合，未定義とします。

(23) 関係演算子－ポインタ
比較対象のポインタで指されているオブジェクトが同一の集成体オブジェクト，または共用体オブジェクトのメンバでない場合，未定義とします。

(24) 単純代入
オブジェクトに格納されている値が，何らかの形でそのオブジェクトの記憶域に重なる他のオブジェクトを通してアクセスされる場合，重なりは完全に一致していなければなりません。さらに，二つのオブジェクトの型は，適合する型の修飾版，または非修飾版でなければなりません。一致しない重なりの代入は，代入によって代入元の値が破壊されます。

(25) 構造体指定子及び共用体指定子
メンバ宣言並びが名前付のメンバを含まない場合，意味を持たない旨の警告メッセージを出力します。ただし，-Xansi オプションを指定した場合は，同一メッセージでエラーとします。

(26) 型修飾子ー const
const 修飾型で定義されたオブジェクトを非 const 修飾型の左辺値を使って変更しようとした場合，未定義とします。

(27) 型修飾子ー volatile
volatile 修飾型で定義されたオブジェクトを，非 volatile 修飾型の左辺値を使って変更しようとした場合，未定義とします。

(28) return 文
式を持たない return 文を実行し，呼び出し元で関数呼び出しの値を使用している場合で，宣言がある場合はメッセージを出力します。宣言がない場合は，関数の戻り値が不定値となります。

(29) 関数定義
可変個引数の実引数を受け付ける関数が，省略記号表記で終わる仮引数型並びをもたずに定義された場合，仮引数の値が不定となります。

(30) 条件付取り込み
置き換え処理によって字句 defined が生成される場合，または defined 単項演算子のマクロ置き換え前の使用法が制約の中で規定した二つの形式のどちらにも一致しない場合，通常の defined として扱います。

(31) マクロ置き換えー前処理句を含まない実引数
実引数が（実引数の置換前に）前処理句を含まない場合，メッセージを出力します。

(32) マクロ置き換えー前処理指令を持つ実引数
実引数の並びの中に，ほかの場合であれば前処理指令として働く前処理字句列がある場合，メッセージを出力します。

(33) # 演算子
置き換えの結果が，正しい単純文字列リテラルでない場合，メッセージを出力します。

(34) ## 演算子
置き換えの結果が，正しい単純文字列リテラルでない場合，メッセージを出力します。

4.1.3 処理系依存

この項では，ANSI 規格における処理系依存項目について説明します。

(1) データ型とサイズ
ワード（4 バイト）中のバイト順序は，“ 下位から上位 ” です。また，符号付き整数は，2 の補数で表現します。最上位ビットには，符号（正，または 0 の場合 0，負の場合 1）が入ります。

- 1 バイト中のビット数は，8 ビットとします。
- オブジェクト中のバイト数，バイト順序，符号化は，次のように規定します。

表 4.1 データ型とサイズ

データ型	サイズ
char	1 バイト
short	2 バイト
int，long，float	4 バイト
double，long double，long long	8 バイト
ポインタ	unsigned int と同じ

(2) 翻訳段階
ANSI 規格では，翻訳における構文規則間の優先順位を 8 つの翻訳段階（翻訳フェーズ）に規定しています。3 段階目の “ 前処理字句と空白類文字の並びへの分割 ” で処理系定義となっている，“ 改行文字以外の空白類文字の空でない並び ” は 1 つに置き換えられずそのまま保持されます。

(3) 診断メッセージ
何らかの構文規則違反，および制約違反を含む翻訳単位に対して，ソース・ファイル名，行番号（特定可能な場合のみ）を含むエラー・メッセージを出力します。なお，エラー・メッセージの書式は “ ワーニング ”，“ アボート・エラー ”，“ フェイタル・エラー ” などに区別されます。出力形式については，「CS+ 統合開発環境 ユーザーズマニュアル メッセージ編」を参照してください。

(4) プログラム開始処理
プログラム開始処理時に呼び出される関数の名前，および型は特に規定しません。
したがって，ユーザ・オウン・コーディング，またはターゲット・システムに依存します。

(5) プログラムの実行
対話型装置の構成については，特に規定しません。
したがって，ユーザ・オウン・コーディング，またはターゲット・システムに依存します。

(6) 文字集合
実行環境文字集合の要素の値は，ASCII コードです。

(7) 多バイト文字
利用できる多バイト文字は，EUC，SJIS，UTF-8，big5，gb2312 です。
コメントと文字列における日本語／中国語記述をサポートしています。

(8) 文字表示の意味
拡張表記の値は，次のように規定します。

表 4.2 拡張表記と意味

拡張表記	値（ASCII）	意味
¥a	07	アラート（警告音）
¥b	08	バックスペース
¥f	0C	フォーム・フィード（改ページ）
¥n	0A	ニュー・ライン（改行）
¥r	0D	キャリッジ・リターン（復帰）
¥t	09	水平タブ
¥v	0B	垂直タブ

(9) 翻訳限界
翻訳に際しての限界値はありません。翻訳可能な最大値は動作するホスト・マシンのメモリに依存します。

(10) 数量的限界

(a) 汎整数型の限界値（limits.h ファイル）
汎整数型（char 型，符号付き／符号なし整数型，および列挙型）で表現できる値の各種限界値を limits.h ファイルに定義しています。
なお，多バイト文字はサポートしていないため，MB_LEN_MAX は該当する限界値を持ちません。そこで，MB_LEN_MAX には 1 として，定義のみ行っています。
次に，limits.h ファイルで定義されている各種限界値を示します。

表 4.3 汎整数型の各種限界値（limits.h ファイル）

名前	値	意味
CHAR_BIT	+8	ビット・フィールドではない最小のオブジェクトのビット数（= 1 バイト）
SCHAR_MIN	-128	signed char 型の最小値
SCHAR_MAX	+127	signed char 型の最大値
UCHAR_MAX	+255	unsigned char 型の最大値
CHAR_MIN	-128	char 型の最小値
CHAR_MAX	+127	char 型の最大値
SHRT_MIN	-32768	short int 型の最小値
SHRT_MAX	+32767	short int 型の最大値
USHRT_MAX	+65535	unsigned short int 型の最大値

名前	値	意味
INT_MIN	-2147483648	int 型の最小値
INT_MAX	+2147483647	int 型の最大値
UINT_MAX	+4294967295	unsigned int 型の最大値
LONG_MIN	-2147483648	long int 型の最小値
LONG_MAX	+2147483647	long int 型の最大値
ULONG_MAX	+4294967295	unsigned long int 型の最大値
LLONG_MIN	-9223372036854775808	long long int 型の最小値
LLONG_MAX	+9223372036854775807	long long int 型の最大値
ULLONG_MAX	+18446744073709551615	unsigned long long int 型の最大値

(b) 浮動小数点型の各種限界値（float.h ファイル）
浮動小数点型の特性に関する各種限界値を float.h ファイルに定義しています。
次に，float.h ファイルで定義されている各種限界値を示します。

表 4.4 浮動小数点型の各種限界値の定義（float.h ファイル）

名前	値	意味
FLT_ROUNDS	+1	浮動小数点加算に対する丸めのモード 1（もっとも近い方向へ丸める）とします。
FLT_RADIX	+2	指数表現の基数（b）
FLT_MANT_DIG	+24	浮動小数点仮数部における FLT_RADIX を底とする数字の数（p）
DBL_MANT_DIG	+53	
LDBL_MANT_DIG	+53	
FLT_DIG	+6	q 桁の 10 進数の浮動小数点数を基数 b の p 桁をもつ浮動小数点数に丸めることができ，再び変更なしに q 桁の 10 進数値に戻すことが可能な 10 進数の桁数（q）
DBL_DIG	+15	
LDBL_DIG	+15	
FLT_MIN_EXP	-125	FLT_RADIX をその値から 1 引いた値でべき乗したとき，正規化された浮動小数点数となるような最小の負の整数 （e_{min}）
DBL_MIN_EXP	-1021	
LDBL_MIN_EXP	-1021	
FLT_MIN_10_EXP	-37	10 をその値でべき乗したとき，正規化された浮動小数点数の範囲内になるような最小の負の整数 $\log_{10} b^{e_{min-1}}$
DBL_MIN_10_EXP	-307	
LDBL_MIN_10_EXP	-307	
FLT_MAX_EXP	+128	FLT_RADIX をその値から 1 引いた値でべき乗したとき，表現可能な有限浮動小数点数となるような最大の整数 （e_{max}）
DBL_MAX_EXP	+1024	
LDBL_MAX_EXP	+1024	
FLT_MAX_10_EXP	+38	10 をその値でべき乗したとき，表現可能な有限浮動小数点数の範囲内になるような最大の整数 $\log_{10} ((1 - b^{-p}) * b^{e_{max}})$
DBL_MAX_10_EXP	+308	
LDBL_MAX_10_EXP	+308	

名前	値	意味
FLT_MAX	3.40282347E + 38F	表現可能な有限浮動小数点数の最大値 $(1 - b^{-p}) * b^{e_{max}}$
DBL_MAX	1.7976931348623158E+308	
LDBL_MAX	1.7976931348623158E+308	
FLT_EPSILON	1.19209290E - 07F	指定された浮動小数点型で表現できる 1.0 と，1.0 より大きい最も小さい値との差異 $b^{1 - p}$
DBL_EPSILON	2.2204460492503131E-016	
LDBL_EPSILON	2.2204460492503131E-016	
FLT_MIN	1.17549435E - 38F	正規化された正の浮動小数点数の最小値 $b^{e_{min} - 1}$
DBL_MIN	2.2250738585072014E-308	
LDBL_MIN	2.2250738585072014E-308	

(11) 識別子
識別子すべてが意味のあるものとして扱います。また，識別子の長さは無制限です。
なお，英字の大文字と小文字は区別されます。

(12) char 型
char 型より大きな量を char 型に格納した場合，char 型に型変換した値となります。
型指定子（signed，unsigned）の付かない単なる char 型は，符号付き整数（signed char）として扱います。

```
char    c = '\777';     /*c の値は -1 となる */
```

(13) 浮動小数点定数
浮動小数点定数は，IEEE754 注に準拠しています。

注 IEEE：Institute of Electrical and Electronics Engineers（電気電信学会）の略称です。
また，IEEE754 とは，浮動小数点演算を扱うシステムにおいて，扱うデータ形式や数値範囲などの仕様の統一化を図った標準です。

(14) 文字定数

(a) ソース・プログラムの文字集合と実行環境における文字集合は，基本的に両者とも ASCII コードで，同一の値をもつメンバと対応します。
ただし，ソース・プログラムにおける文字列には，日本語文字コードが使用できます（「(8) 文字表示の意味」を参照）。

(b) 2 つ以上の文字を含む整数文字定数の値は，最後の 1 文字が有効値となります。

(c) 基本的な実行環境文字集合で表現されない文字やエスケープ・シーケンスを含む場合，次のようになります。

<1> 8 進数エスケープ・シーケンス，および 16 進数エスケープ・シーケンスは，その 8 進数表記，および 16 進数表記で示される値となります。

\777	511

<2> 単純エスケープ・シーケンスは，次のようになります。

\'	'
\"	"
\?	?
\\	\

<3> ¥a，¥b，¥f，¥n，¥r，¥t，¥v については，「(8) 文字表示の意味」で示されている値と同値になります。

(d) 多バイト文字の文字定数はサポートしていません。

(15) 文字列
文字列中に日本語／中国語が記述できます。
デフォルトの文字コードは，シフト JIS となります。

入力ソース・ファイルの中の文字コードは，CC-RH の -Xcharacter_set オプションで選択できます。

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

(16) ヘッダ・ファイル名
ヘッダ・ファイル名の 2 つの形式（< >，" "）内の列を，ヘッダ・ファイル，または外部ソース・ファイル名に反映する方法は，「(33) ヘッダ・ファイル取り込み」で規定します。

(17) コメント
コメント中に日本語／中国語が記述できます。文字コードは，「(15) 文字列」の場合と同じです。

(18) 符号付き定数と符号なし定数
汎整数型の値がよりサイズの小さい符号付き整数に変換される場合，上位ビットを切り捨てて，ビット列イメージをコピーします。
また，符号なし整数が，対応する符号付き整数に変換される場合，内部表現は変化しません。

(19) 浮動小数点と汎整数
汎整数型の値が浮動小数点型に型変換される際，型変換される値が，表現しうる値の範囲内にはあるが正確に表現することができない場合，その結果は，表現しうる最も近い値へ丸められます。

(20) double 型と float 型
double 型を float 型に変換する場合，または long double 型を double 型，または float 型に変換するとき，型変換される値が，表現しうる値の範囲内にはあるが正確に表現することができない場合，その結果は，表現しうる最も近い値へ丸められます。

(21) ビット単位の演算子における符号付き型
ビット単位の演算子における符号付き型に対する特性は，シフト演算子については，「(27) ビット単位のシフト演算子」の規定に準じます。
また，その他の演算子については，符号なしの値として（ビット・イメージのままで）計算するものとします。

(22) 構造体と共用体のメンバ
共用体のメンバの値がそれと異なるメンバに格納される場合，整列条件にしたがって格納されるため，その共用体のあるメンバへのアクセスは，後からアクセスした型としてアクセスされます。

(23) sizeof 演算子
“sizeof” 演算子の結果は，「(1) データ型とサイズ」におけるオブジェクト中のバイトに関する規定に準じます。
なお，構造体と共用体については，パディング領域を含んだバイト数とします。

(24) キャスト演算子
ポインタを汎整数型に変換する場合，要求される変数のサイズは，unsigned long 型と同じサイズです。変換結果は，ビット列がそのまま保存されます。
また，任意の整数はポインタに型変換できますが，整数をポインタに型変換した場合，整数のビットパターンがそのまま保持されます。

(25) 乗除／剰余演算子
整数同士の除算で割り切れず，オペランドが負の値をもつ場合，“/” 演算子の結果は，除数，または被除数のいずれか一方が負の場合は，代数的な商よりも大きい最小の整数となります。
ただし，どちらも負の場合は，代数的な商よりも小さい最大の整数となります。
また，オペランドが負の値をもつ場合，“%” 演算子の結果の符号は第 1 オペランドの符号とします。

(26) 加減演算子
同一配列の要素を指す 2 つのポインタが減算される場合，結果の型は unsigned long 型となります。

(27) ビット単位のシフト演算子
“E1 >> E2” において，E1 が符号付きの型で負の値をもつ場合，算術シフトを行います。

(28) 記憶域クラス指定子
記憶域クラス指定子 “register” の宣言の有無にかかわらず，可能なかぎり高速にアクセスするように最適化を行います。

(29) 構造体と共用体指定子

(a) signed，unsigned の付かない単なる int 型ビット・フィールドは，符号付きとして扱い，最上位ビットは符号ビットとして扱います。

(b) ビット・フィールドを保持するために，十分な大きさの任意のアドレス付け可能な記憶域単位を割り付けることができますが，十分な領域がなかった場合，合わなかったビット・フィールドはフィールドの型の整列条件に合わせて次の単位に詰め込まれます。
単位内のビット・フィールドの割り付け順序は下位から上位です。

(c) 1 つの構造体，または共用体の非ビット・フィールドの各メンバは，次のように境界整列されます。

char，unsignd char 型，およびその配列	バイト境界
short，unsigned short 型，およびその配列	2 バイト境界
その他（ポインタを含む）	4 バイト境界

ただし，パッキングをした場合は，すべてのメンバの境界はパッキングの指定値に依存します。

(30) 列挙型指定子
列挙型の型は，signed int 型とします。
ただし，-Xenum_type=auto オプション指定時は，各列挙型について，その型のすべての列挙子の値を表現可能な最小の整数型として扱います。

(31) 型修飾子
“volatile” 修飾された型をもつデータへのアクセスは，データがマッピングされているアドレス（I/O ポートなど）に依存します。ただし，同変数へのアクセス箇所への最適化は行われません。

(32) 条件組み込み

(a) 条件組み込みで指定される文字定数に対する値と，その他の式中に現れる文字定数の値とは等しくなります。

(b) 単一文字の文字定数は，負の値を持ちません。

(33) ヘッダ・ファイル取り込み

(a) “#include < 文字列 >” という形式の前処理指示
“#include < 文字列 >” という形式の前処理指示は，“ 文字列 ” がフルパスのファイル名でない場合注，指定されたフォルダ（-I オプション）からヘッダ・ファイルを検索し，次に標準インクルード・ファイル・フォルダ（ccrh が置かれた bin フォルダからの相対パスでの ..￥inc フォルダ）を検索します。
なお，“<” と “>” の区切り記号の間に指定された文字列で一意に識別されるヘッダ・ファイルを探し出すと，そのヘッダ・ファイルの内容全体で置き換えます。

注 “￥” と “/” の両者がフォルダの区切りとしてみなされます。

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

検索順は，次のとおりです。

- -I で指定したフォルダ
- 標準インクルード・ファイル・フォルダ

(b) “#include " 文字列 "” という形式の前処理指示
“#include " 文字列 "” という形式の前処理指示は，“ 文字列 ” がフルパスのファイル名でない場合注，ソース・ファイルがあるフォルダからヘッダ・ファイルを検索し，次に，指定したフォルダ（-I オプション），最後に標準インクルード・ファイル・フォルダ（ccrh が置かれた bin フォルダからの相対パスでの ..￥inc フォルダ）を検索します。
なお，“ " ”“ " ” の区切り記号の間に指定された文字列で一意に識別されるヘッダ・ファイルを探し出すと，そのヘッダ・ファイルの内容全体で置き換えます。

注 “￥” と “/” の両者がフォルダの区切りとしてみなされます。

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

検索順は，次のとおりです。

- ソース・ファイルがあるフォルダ
- -I で指定したフォルダ
- 標準インクルード・ファイル・フォルダ

(c) “#include 前処理字句列 ” という形式
“#include 前処理字句列 ” という形式において，前処理字句列が単一で < 文字列 >，または " 文字列 " の形式に置換されるマクロである場合にのみ，単一のヘッダ・ファイル名の前処理字句として扱われます。

(d) “#include < 文字列 >” という形式の前処理指示
（最終的に）区切られた列とヘッダ・ファイル名との間においては，列中の英文字の長さを判別し，

コンパイラ動作環境において有効なファイル名長までが有効

となります。ファイルを探すフォルダについては，上記の規定に準じます。

(34) #pragma 指令

CC-RH では，次の #pragma 指令が指定できます。

(a) データ／プログラムのメモリ割り当て

```
#pragma section セクション種別    [" セクション名 "]
```

変数を任意のセクションに割り当てます。
なお，配置方法についての詳細は「4.2.4.1 データとプログラムのセクション割り当て」を参照してください。

(b) アセンブラ命令の記述

```
#pragma inline_asm [(] 関数名 [(size= 数値 )][, 関数名 [(size= 数値 )], ...][)]
```

C 言語中に，アセンブラ命令を記述することができます。
なお，記述方法についての詳細は「4.2.4.2 アセンブラ命令の記述」を参照してください。

(c) インライン展開指定

```
#pragma inline   [(] 関数名 [, 関数名 , ...][)]
#pragma noinline [(] 関数名 [, 関数名 , ...][)]
```

インライン展開する関数を指定することができます。
なお，インライン展開についての詳細は「4.2.4.3 インライン展開」を参照してください。

(d) 割り込み／例外ハンドラ指定

```
#pragma interrupt   [(] 関数名 [( 割り込み仕様 [, 割り込み仕様 ]...)] [,...] [)]
```

割り込み／例外処理ハンドラを C 言語で記述します。
なお，記述方法についての詳細は「(3) 割り込み／例外ハンドラの記述方法」を参照してください。

(e) 割り込み禁止関数指定

```
#pragma block_interrupt [(] 関数名 [)]
```

関数全体を割り込み禁止にします。
なお，記述方法についての詳細は「(2) 関数全体の割り込みを禁止する方法」を参照してください。

(f) 構造体パッキング指定

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

構造体パッキングを指定します。数値はパッキング値，すなわちメンバのアライメント値を指定します。数値には 1，2，4 が指定できます。数値を指定しない場合，デフォルトのアライメントとなります。
なお，記述方法についての詳細は「4.2.4.8 構造体パッキング」を参照してください。

(g) ビット・フィールドの割り付け

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

ビット・フィールドの並び順の切り替えを指定します。
なお，記述方法については「4.2.4.9 ビット・フィールドの割り付け」を参照してください。

(h) コア番号指定（マルチコア用）

```
#pragma pmodule pm 指定
```

コア番号指定を行う関数を指定することができます。
なお，コア番号指定についての詳細は「4.2.4.10 コア番号指定（マルチコア用）」を参照してください。

(35) あらかじめ定義されたマクロ名

以下に，サポートしているマクロ名を示します。

表 4.5 サポートしているマクロ

マクロ名	定義
__LINE__	その時点でのソース行の行番号（10 進数）。
__FILE__	ソース・ファイルの名前（文字列定数）。
__DATE__	ソース・ファイルの翻訳日付（“Mmm dd yyyy” の形式をもつ文字列定数。ここで，月の名前は ANSI 規格で規定されている asctime 関数で生成されるもの（英字 3 文字の並びで最初の 1 文字のみ大文字）と同じもの。dd の最初の文字は値が 10 より小さい場合空白とします）。
__TIME__	ソース・ファイルの翻訳時間（asctime 関数で生成される時間と同じような “hh : mm : ss” の型式をもつ文字列定数）。
__STDC__	10 進定数 1（-Xansi オプション指定時に定義）。注
__RENESAS__	値は設定されません。
__RENESAS_VERSION__	バージョンが V.XX.YY.ZZ の場合，0xXXYYZZ00 とします。 例）V.1.00.00 → -D__RENESAS_VERSION__=0x01000000
__CCRH__	値は設定されません。
__CCRH	値は設定されません。
__RH850__	値は設定されません。
__RH850	値は設定されません。
__v800 __v800__	10 進定数 1。
__v850 __v850__	10 進定数 1。
__v850e __v850e__	10 進定数 1（CX で，ターゲット・デバイスに V850E*x* を指定した場合に定義）。
__v850e2 __v850e2__	10 進定数 1（CX で，ターゲット・デバイスに V850E2 を指定した場合に定義）。
__v850e2v3 __v850e2v3__	10 進定数 1（CX で，ターゲット・デバイスに V850E2V3 のインストラクション・セットをもつデバイスを指定した場合に定義）。
__v850e3v5__	値は設定されません（-Xcommon オプションで，v850e3v5，または rh850 を指定した場合に定義）。
__v850e3v5	値は設定されません（-Xcommon オプションで，v850e3v5，または rh850 を指定した場合に定義）。
__K0R __K0R__	10 進定数 1（CX で，ターゲット・デバイスに 78K0R を指定した場合に定義）。
__CX __CX__	10 進定数 1。
__DBL4	-Xdbl_size=4 指定時のみ 値は設定しない
__DBL8	-Xdbl_size=8 指定時または -Xdbl_size 指定がない場合 値は設定しない
__DOUBLE_IS_64BITS__	-Xdbl_size=8 指定時または -Xdbl_size 指定がない場合 値は設定しない
__DOUBLE_IS_32BITS__	-Xdbl_size=4 指定時のみ 値は設定しない

マクロ名	定義
__RON	-Xround=nearest 指定時のみ 値は設定されません
__ROZ	-Xround=zero 指定時のみ 値は設定されません
__BITLEFT	値は設定されません（-Xbit_order オプションで，left を指定した場合に定義）。
__BITRIGHT	値は設定されません（-Xbit_order オプションで，right を指定した場合に定義）。
__AUTO_ENUM	値は設定されません（-Xenum_type オプションで，auto を指定した場合に定義）。
__FPU	値は設定されません（-Xfloat オプションで，fpu を指定した場合に定義）。
__CHAR_SIGNED__	値は設定されません。
__CHAR_UNSIGNED__	10 進定数 1（-Xchar オプションで，符号なしを指定した場合に定義）。
__CPU マクロ _ __CPU マクロ __	ターゲット CPU を示すマクロで 10 進定数 1。 デバイス・ファイル中の「品種指定名」で示される文字列の先頭に “__”，末尾に “_”，または “__” を付けたものが定義されます。
レジスタ・モード・マクロ	ターゲット CPU を示すマクロで値は設定されません。 レジスタ・モードと定義されるマクロは次のとおりです。 32 レジスタ・モード ： __reg32__ 22 レジスタ・モード ： __reg22__ 汎用レジスタ・モード ： __reg_common__
_LIT	値は設定されません。
__MULTI_LEVEL__	-Xmulti_level=level で指定した level 値
_FLT	値は設定されません。
__FLT__	値は設定されません。
_FPS	値は設定されません。
_FPD	値は設定されません。
__MULTI_CORE__	10 進定数 1（-Xmulti オプションを指定した場合に定義）。
__MULTI_CMN__ __MULTI_PE*n*__	10 進定数 1（-Xmulti オプションによるコア種別を指定した場合に定義（*n* は数値））。

注 -Xansi オプション指定時の処理については，「4.1.5 ANSI 規格に厳密な処理オプション」を参照してください。

4.1.4 C99 言語機能

この項では，CC-RH がサポートする C99 言語機能について説明します。

(1) // によるコメント
// （スラッシュ 2 つ）より始まり改行までをコメントとします。改行の直前の文字が￥の場合，次の行も続いた 1 つのコメントとします。

(2) ワイド文字列の結合
文字列定数とワイド文字列定数を結合した場合，ワイド文字列定数となります。

(3) _Bool 型
_Bool 型をサポートします。

(4) long long int 型

long long int 型をサポートします。long long int 型は 8 バイトの整数型です。
定数値の末尾 LL もサポートします。ビット・フィールドの型にも指定可能です。

(5) 汎整数拡張
_Bool 型と long long 型のサポートに伴って，汎整数拡張も C99 の仕様に従います。

(6) 既定の実引数拡張
_Bool 型と long long 型のサポートに伴って，既定の実引数拡張も C99 の仕様に従います。

- _Bool 型は int 型（4 バイト）に拡張して関数を呼び出します。
- (unsigned) long long 型は，8 バイトのまま関数を呼び出します。

(7) enum 定義の最後の列挙子の後のカンマ許可
enum 型を定義する際，列挙子の列挙の最後の ”，（カンマ）” を許可します。

```
enum EE {a, b, c,};
```

(8) inline キーワード（inline 関数）
inline キーワードをサポートします。
また，以下の書式による pragma 指令でも指定できます。

```
#pragma inline  [(] 関数名 [, 関数名 , ...][)]
```

なお，インライン展開についての詳細は「4.2.4.3 インライン展開」を参照してください。

(9) 整数定数の型
long long 型の追加に伴って，整数定数の型は次のとおりになります。
整数定数の型は，次の並びのうちでその値を表現できる最初の型となります。

表 4.6 整数定数の型（long long 型が有効な場合（-Xansi 指定なし））

接尾語	10 進定数	8 進定数，または 16 進数
なし	int long int unsigned long int 注 long long int	int unsigned int long int unsigned long int long long int unsigned long long int
u，または U	unsigned int unsigned long int unsigned long long int	unsigned int unsigned long int unsigned long long int
l，または L	long int long long int	long int unsigned long int long long int unsigned long long int
u，または U，および l，または L の両方	unsigned long int unsigned long long int	unsigned long int unsigned long long int
ll，または LL	long long int	long long int unsigned long long int
u，または U，および ll，または LL の両方	unsigned long long int	unsigned long long int

注 C99 の仕様と異なります。

表 4.7 整数定数の型（long long 型が無効な場合（-Xansi 指定あり））

接尾語	10 進定数	8 進定数，または 16 進数
なし	int long int unsigned long int	int unsigned int long int unsigned long int

接尾語	10 進定数	8 進定数，または 16 進数
u，または U	unsigned int unsigned long int	unsigned int unsigned long int
l，または L	long int	long int unsigned long int
u，または U，および l，または L の両方	unsigned long int	unsigned long int

4.1.5 ANSI 規格に厳密な処理オプション

CC-RH で -Xansi オプションを指定した場合，ANSI 規格に厳密な処理が行われます。
次に，-Xansi オプションを指定した場合と，指定しない場合の処理の違いを示します。

表 4.8 言語仕様に厳密な -Xansi オプション指定時の処理

項目	-Xansi 指定あり	-Xansi 指定なし
トライグラフ系列	トライグラフ系列の置換を行います。	置換しません。
ビット・フィールド	ビット・フィールドに int 型以外の型を指定した場合，エラー[注 1] とします。	int 型以外の型の指定を許可します。
引数のスコープ	関数の引数と同名の自動変数を宣言した場合，二重定義のエラーとします。	警告メッセージを出力し，自動変数を有効とします。
ポインタの代入 1	汎整数型[注 2] 変数へのポインタ型の数値の代入は，エラーとします。	警告メッセージを出力し，キャストして代入します。
ポインタの代入 2	異なる型を指すポインタ同士の代入は，エラーとします。	警告メッセージを出力し，許可します。
型変換	左辺値でない配列のポインタへの変換は，エラーとします。	警告メッセージを出力し，許可します。
比較演算子	算術型変数とポインタの比較は，エラーとします。	警告メッセージを出力し，許可します。
条件演算子	第 2 式と第 3 式がともに汎整数型，同じ構造体，同じ共用体，または代入先と同じ型へのポインタ型の数値のいずれでもない場合，エラーとします。	警告メッセージを出力し，キャストして代入します。
# 行番号	エラーとします。	“#line” 行番号と同様に扱います。[注 2]
行の途中の # 文字	“#” 文字が行の途中で現れた場合，エラーとします。	警告メッセージを出力し，許可します。
#pragma inline された関数の引数	指定した関数の呼び出しと定義の間で戻り値の型や引数の型が異なるが型変換が可能である場合，エラーとします。	戻り値の型は呼び出し側の型に，引数は関数定義での型に変換して，インライン展開します。
_Bool 型 long long 型	エラーとします。	指定を許可します。
構造体指定子 共用体指定子	メンバ宣言並びが名前付のメンバを含まない場合，意味を持たない旨のエラーメッセージを出力します。	メンバ宣言並びが名前付のメンバを含まない場合，意味を持たない旨の警告メッセージを出力します。
_asm	警告メッセージを出力し，関数呼び出しとして扱います。 ただし，__asm は有効とします。	アセンブラ挿入[注 4] として扱います。
__STDC__	値が 1 のマクロとして定義します。	定義しません。

項目	-Xansi 指定あり	-Xansi 指定なし
2 進定数	“0b”，または “0B” と，その後ろに続く 1 個以上の “0”，または “1” の数字の並びをエラーとします。	“0b”，または “0B” と，その後ろに続く 1 個以上の “0”，または “1” の数字の並びを 2 進定数とします。

注 1. “E” で始まる通常のエラー。以下同じです。

注 2. ANSI 規格を参照してください。

4.1.6 データの内部表現と領域

この項では，CC-RH が扱うデータのそれぞれの型における，内部表現と値域について説明します。

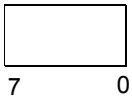
(1) 整数型

(a) 内部表現

領域の左端ビットは，符号付きの型（“unsigned” を伴わずに宣言された型）では，符号ビットとなります。符号付きの型において，値は 2 の補数表現で表されます。

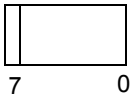
図 4.1 整数型の内部表現

_Bool

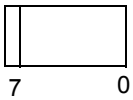


ゼロビット目のみが意味を持ちます。1 ～ 7 ビット目は不定値となります。
-Xansi オプション使用時は，_Bool 型は C90 違反でエラーとなります。

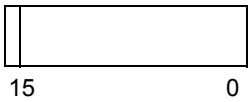
char



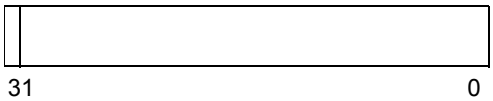
signed char（unsigned では符号ビットなし）



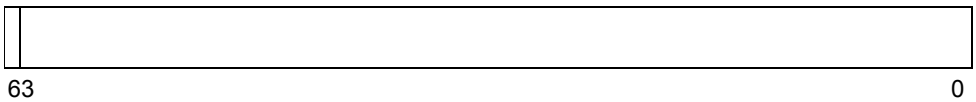
short（unsigned では符号ビットなし）



int, long（unsigned では符号ビットなし）



long long（unsigned では符号ビットなし）



-Xansi オプション使用時は，long long 型は C90 違反でエラーとなります。

(b) 値域

表 4.9 整数型の値域

型	値域
char	-128 ～ +127
short	-32768 ～ +32767
int	-2147483648 ～ +2147483647

型	値域
long	-2147483648 ～ +2147483647
long long	-9223372036854775808 ～ +9223372036854775807
unsigned char	0 ～ 255
unsigned short	0 ～ 65535
unsigned int	0 ～ 4294967295
unsigned long	0 ～ 4294967295
unsigned long long	0 ～ 18446744073709551615

(2)　浮動小数点型

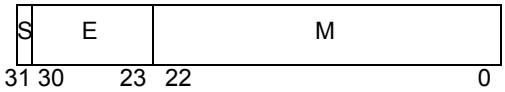
(a)　内部表現

浮動小数点型データの内部表現は，IEEE754 注に準拠しています。領域の左端のビットは，符号ビットとなります。この符号ビットの値が 0 であれば正の値に，1 であれば負の値になります。

注　　IEEE：Institute of Electrical and Electronics Engineers（電気電信学会）の略称です。
また，IEEE754 とは，浮動小数点演算を扱うシステムにおいて，扱うデータ形式や数値範囲などの仕様の統一化を図った標準です。

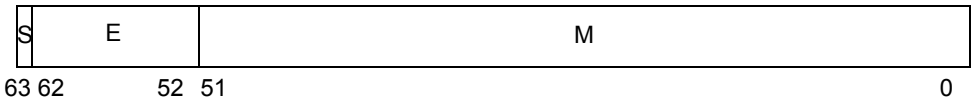
図 4.2　　浮動小数点型の内部表現

float



S：　仮数部の符号ビット
E：　指数部（8 ビット）
M：　仮数部（23 ビット）

double, long double



S：　仮数部の符号ビット
E：　指数部（11 ビット）
M：　仮数部（52 ビット）

(b)　値域

表 4.10　　浮動小数点型の値域

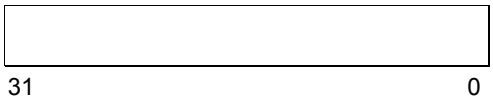
型	値域
float	1.17549435E-38F ～ 3.40282347E+38F
double	2.2250738585072014E-308 ～ 1.7976931348623158E+308
long double	2.2250738585072014E-308 ～ 1.7976931348623158E+308

(3)　ポインタ型

(a)　内部表現

ポインタ型の内部表現は，unsigned int 型の内部表現と同じです。

図 4.3 ポインタ型の内部表現

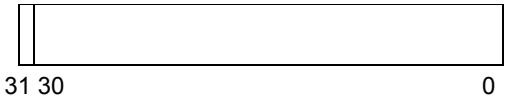


(4) 列挙型

(a) 内部表現

列挙型の内部表現は，singed int 型の内部表現と同じです。領域の左端のビットは，符号ビットとなります。

図 4.4 列挙型の内部表現



-Xenum_type=auto オプション指定時には，「(30) 列挙型指定子」を参照してください。

(5) 配列型

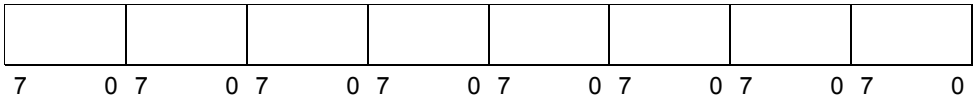
(a) 内部表現

配列型の内部表現は，配列の要素を，その要素の整列条件（alignment）を満たす形で並べたものとなります。

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

上記の例に示した配列に対する内部表現は，次のようになります。

図 4.5 配列型の内部表現



(6) 構造体型

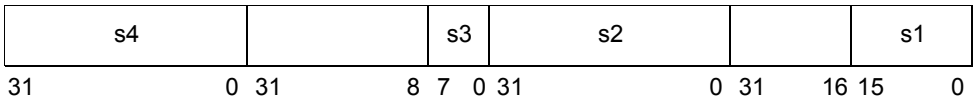
(a) 内部表現

構造体型の内部表現は，構造体の要素をその要素の整列条件を満たす形で並べたものとなります。

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

この例に示した構造体に対する内部表現は，次のようになります。

図 4.6 構造体型の内部表現



なお，構造体パッキング機能利用時の内部表現は，「4.2.4.8 構造体パッキング」を参照してください。

(7) 共用体型

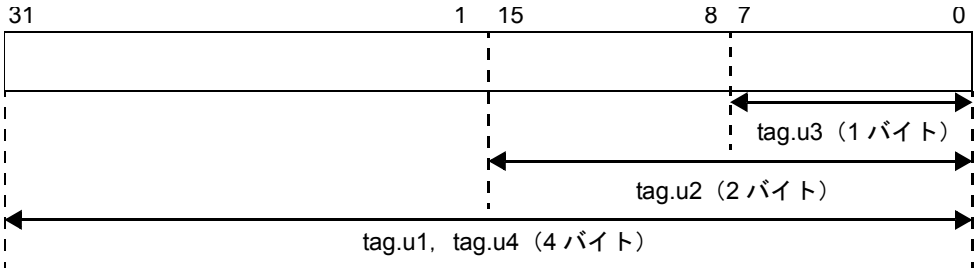
(a) 内部表現

共用体はそのメンバがすべてオフセット 0 から始まり，そのメンバの任意のものを収容するのに十分なサイズを持つ構造体と考えられます。つまり，共用体型の内部表現は，同じアドレスに共用体の要素それぞれが単体で置かれているのと同様です。

```
union {
    int     u1;
    short   u2;
    char    u3;
    long    u4;
} tag;
```

この例に示した共用体に対する内部表現は，次のようになります。

図 4.7 共用体型の内部表現



(8) ビット・フィールド

(a) 内部表現

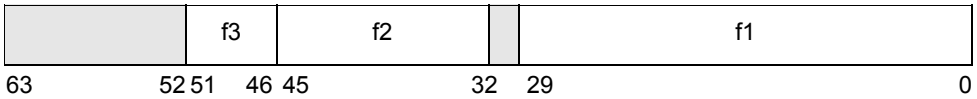
ビット・フィールドに対しては，宣言された数のビットを含む領域が取られます。符号付きの型として，または符号を明示せずに宣言されたビット・フィールドに対しては，最上位ビットは符号ビットとなります。
最初に宣言されたビット・フィールドは，ビット・フィールド宣言時の型のサイズの領域の最下位ビットから割り当てられます。ビット・フィールドに対し，その前のビット・フィールドに続けて領域を割り当てると，その領域がそのビット・フィールドの宣言において指定された型の整列条件を満たす境界を越えてしまう場合，そのビット・フィールドに対する領域はその整列条件を満たしている境界から割り当てられます。
-Xbit_order=left オプション，または #pragma bit_order left を指定することで，ビット・フィールド・メンバを上位ビット側から割り当てることも可能です。詳細は，「4.2.4.9 ビット・フィールドの割り付け」を参照してください。

例 1.

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

この例に示したビット・フィールドに対する内部表現は，次のようになります

図 4.8 ビット・フィールドの内部表現

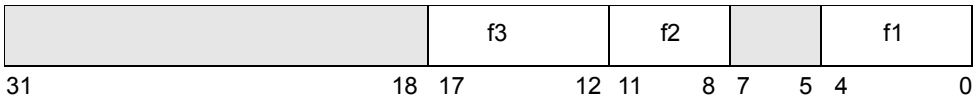


例 2.

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

この例に示したビット・フィールドに対する内部表現は，次のようになります

図 4.9 ビット・フィールドの内部表現



ANSI 規格ではビット・フィールドに int 型，および unsigned int 型のみ指定できますが，CC-RH では，ビット・フィールドに char 型，short 型，long 型，long long 型，およびそれらの unsigned 型も指定することができます。
なお，構造体パッキング機能利用時のビット・フィールドの内部表現は，「4.2.4.8 構造体パッキング」を参照してください。

(9) 整列条件

(a) 基本型に対する整列条件
次に，基本型に対する整列条件を示します。
ただし，CC-RH の -Xinline_strcpy を指定した場合，配列型はすべて 4 バイト境界となります。

表 4.11 基本型に対する整列条件

基本型	整列条件
（unsigned）char とその配列型 _Bool 型	バイト境界
（unsigned）short とその配列型	2 バイト境界
（ポインタを含む）その他の基本型 (unsigned) long long とその配列型 double とその配列型	4 バイト境界
long double とその配列型	4 バイト境界

(b) 共用体型に対する整列条件
共用体型に対する整列条件は，構造体を構成するメンバのうち，最大の整列条件をもつ型の整列条件と同じになります。

それぞれの場合における例を示します。

例 1.

```
union   tug1 {
    unsigned short  i;  /*2 バイト・メンバ */
    unsigned char   c;  /*1 バイト・メンバ */
};  /* 共用体は 2 バイトで整列 */
```

例 2.

```
union   tug2 {
    unsigned int    i;  /*4 バイト・メンバ */
    unsigned char   c;  /*1 バイト・メンバ */
};  /* 共用体は 4 バイトで整列 */
```

(c) 構造体型に対する整列条件
構造体型に対する整列条件は，構造体を構成するメンバのうち，最大の整列条件をもつ型の整列条件と同じになります。

それぞれの場合における例を示します。

例 1.

```
struct  ST {
    char    c;      /*1 バイト・メンバ */
};  /* 構造体は 1 バイトで整列 */
```

例 2.

```
struct  ST {
    char    c;      /*1 バイト・メンバ */
    short   s;      /*2 バイト・メンバ */
};  /* 構造体は 2 バイトで整列 */
```

例 3.

```
struct  ST {
    char    c;      /*1 バイト・メンバ */
    short   s;      /*2 バイト・メンバ */
    short   s2;     /*2 バイト・メンバ */
};  /* 構造体は 2 バイトで整列 */
```

例 4.

```
struct  ST {
    char    c;      /*1 バイト・メンバ */
    short   s;      /*2 バイト・メンバ */
    int     i;      /*4 バイト・メンバ */
};  /* 構造体は 4 バイトで整列 */
```

例 5.

```
struct  ST {
    char        c;      /*1 バイト・メンバ */
    short       s;      /*2 バイト・メンバ */
    int         i;      /*4 バイト・メンバ */
    long long   ll;     /*4 バイト・メンバ */
};  /* 構造体は 4 バイトで整列 */
```

(d) 関数引数に対する整列条件
関数引数に対する整列条件は，4 バイト境界となります。

(e) 実行プログラムに対する整列条件
リロケータブルなオブジェクト・ファイルをリンクして実行可能なオブジェクト・ファイルを生成する際の整列条件は，2 バイト境界となります。

4.1.7 セクション名

以下に，予約されているセクションの名前とそれらのセクション・タイプ，およびセクション属性を示します。

表 4.12 予約セクション

セクションの種類	デフォルト・セクション名	内容	整列サイズ
プログラム領域	.text	実行プログラム	2

セクションの種類	デフォルト・セクション名	内容	整列サイズ
初期値なしデータ領域	.bss	r0 相対の ld/st（2 命令）でアクセスするセクション	4
	.zbss	r0 相対の ld/st 命令でアクセス可能なセクション #pragma section r0_disp* 指定時に生成	
	.zbss23		
	.ebss	ep 相対の ld/st 命令でアクセス可能なセクション #pragma section ep_disp* 指定時に生成	
	.ebss23		
	.tbss4	ep 相対の sld/sst 命令でアクセス可能なセクション #pragma section ep_disp* 指定時に生成	
	.tbss5		
	.tbss7		
	.tbss8		
	.sbss	gp 相対の ld/st 命令でアクセス可能なセクション #pragma section gp_disp* 指定時に生成	
	.sbss23		
初期値ありデータ領域	.data	r0 相対の ld/st（2 命令）でアクセスするセクション	4
	.zdata	r0 相対の ld/st 命令でアクセス可能なセクション #pragma section r0_disp* 指定時に生成	
	.zdata23		
	.edata	ep 相対の ld/st 命令でアクセス可能なセクション #pragma section ep_disp* 指定時に生成	
	.edata23		
	.tdata4	ep 相対の sld/sst 命令でアクセス可能なセクション #pragma section ep_disp* 指定時に生成	
	.tdata5		
	.tdata7		
	.tdata8		
	.sdata	gp 相対の ld/st 命令でアクセス可能なセクション #pragma section gp_disp* 指定時に生成	
	.sdata23		
定数データ領域	.const	r0 相対の ld/st（2 命令）でアクセスするセクション	4
	.zconst	r0 相対の ld/st 命令でアクセス可能なセクション #pragma section .zconst* 指定時に生成	
	.zconst23		

デフォルト・セクション名は #pragma section で変更することができます。

4.1.8 レジスタ・モード

CC-RH では，3 つのレジスタ・モードが提供されています。レジスタ・モードを効率的に指定することにより，割り込み処理時やタスク切り替え時に，一部のレジスタの退避／復帰処理が不要となり，処理速度が高められます。レジスタ・モードの指定は，CC-RH のレジスタ・モード指定オプション（-Xreg_mode）によって行います。この機能は，CC-RH が内部で使用するレジスタの本数を段階的に抑制し，次の効果が期待できます。

- 余ったレジスタをアプリケーション・プログラム（アセンブラ・ソース・プログラム）で自由に使うことができる。
- 退避，復帰で生じるオーバーヘッドが減少する。

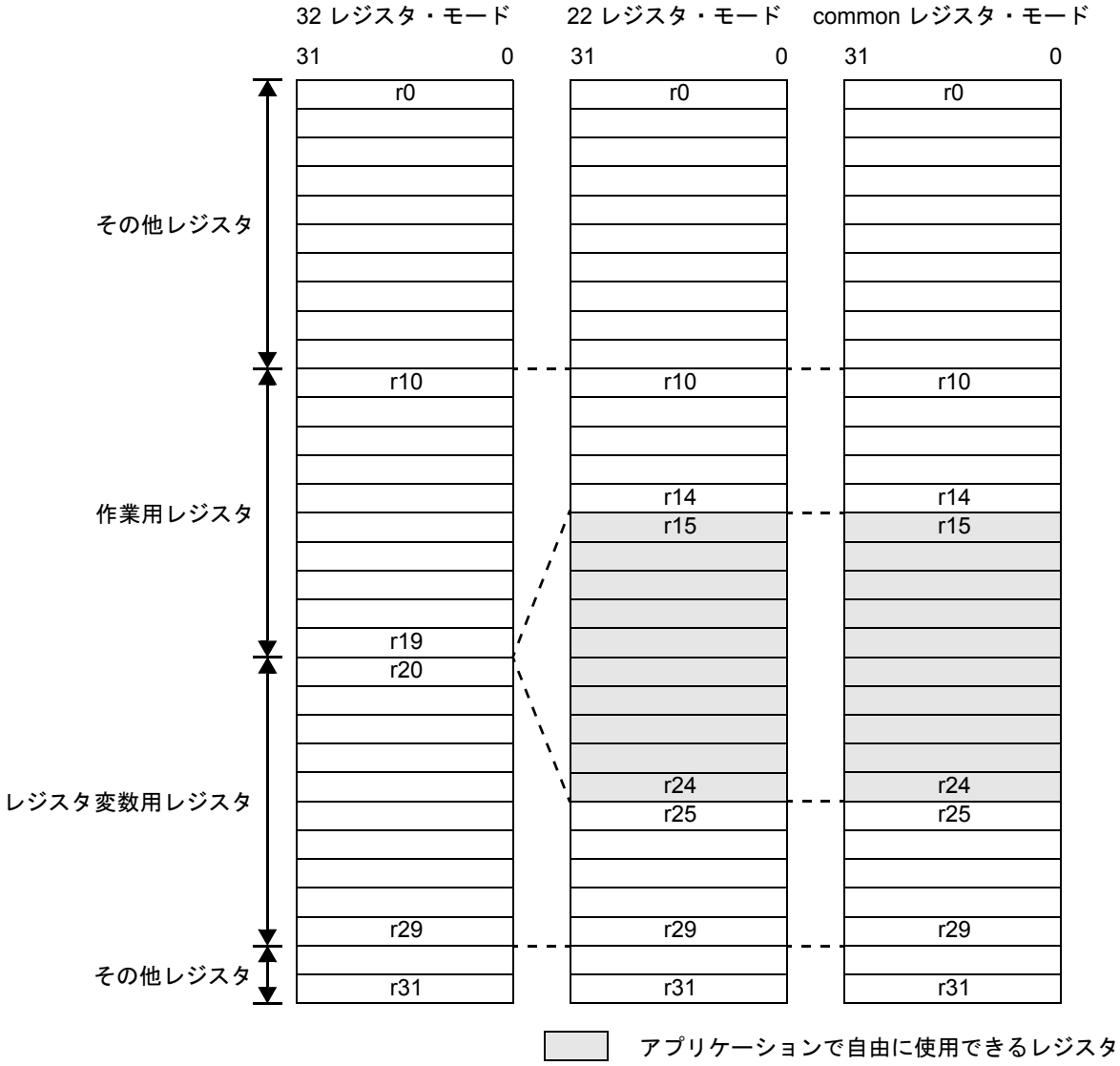
注意 CC-RH によるレジスタ割り付けの対象となる変数の多いアプリケーション・プログラムでは，レジスタ・モードの指定によって，それまでレジスタに割り付けられていた変数がメモリ・アクセスとなり，その分，処理速度が低下することがあります。

次表，および次図に，CC-RH のレジスタ・モードとして提供されている 3 つのモードを示します。

表 4.13 CC-RH が提供するレジスタ・モード

レジスタ・モード	作業用レジスタ	レジスタ変数用レジスタ
32 レジスタ・モード（デフォルト）	r10 ～ r19	r20 ～ r29
22 レジスタ・モード	r10 ～ r14	r25 ～ r29
common レジスタ・モード	r10 ～ r14	r25 ～ r29

図 4.10 レジスタ・モードと使用可能レジスタ



コマンド・ラインにおける指定例

```
> ccrh -Xreg_mode=22 file.c    ← 22 レジスタ・モードでコンパイル
```

4.2 拡張言語仕様

この節では，CC-RH で独自に追加されている拡張言語仕様について説明します。

4.2.1 予約語

CC-RH では，拡張機能を実現するために次の字句を予約語として追加しています。これらの字句も ANSI-C のキーワードと同様，ラベルや変数名として使用することはできません。
次に，CC-RH で追加されている予約語一覧を示します。

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

また，2 個の下線 (__) を含む名前も ANSI-C のキーワード同様，ラベル名や変数名として使用することはできません。

4.2.2 コンパイラ生成シンボル

次に，コンパイラが内部処理で利用するために生成するシンボルの一覧を示します。
以下のシンボルと同名のシンボルは利用できません。

表 4.14 コンパイラ生成シンボル

対象	生成シンボル
extern 関数名	_ 関数名
static 関数名	_ 関数名 *.num* 注
extern 変数名	_ 変数名
ファイル内 static 変数	_ 変数名 *.num* 注
関数内 static 変数	_ 変数名 *.num* 注 . 関数ラベル
関数内ラベル	.BB.LABEL.*num1_num2* 注
文字列	.STR.*num*
switch テーブル	.SWITCH.LABEL.*num1_num2* 注 .SWITCH.LABEL.*num1_num2* 注 .END

注 *num*，*num1*，*num2* は任意の数字です。

4.2.3 #pragma 指令

次に，拡張言語仕様としてサポートしている #pragma 指令を示します。

表 4.15 サポートしている #pragma 指令

#pragma 指令	内容
#pragma section 属性指定文字 " セクション名 "	データとプログラムのセクション割り当て
#pragma section 属性指定文字	
#pragma section [文字列]	
#pragma inline_asm [(] 関数名 [(size= 数値)][, 関数名 [(size= 数値)], ...][)]	アセンブラ命令の記述
#pragma inline [(] 関数名 [, 関数名 , ...][)]	インライン展開
#pragma noinline [(] 関数名 [, 関数名 , ...][)]	
`#pragma interrupt [(] 関数名 [( 割り込み仕様 [, 割り込み仕様 ]...)] [,...] [)]`	割り込み／例外ハンドラ指定

#pragma 指令	内容
#pragma block_interrupt [(] 関数名 [)]	割り込み禁止関数指定
#pragma rtos_task [関数名]	タスク指定
#pragma pack [(][1|2|4][)]	構造体パッキング指定
#pragma bit_order [{left|right}]	ビット・フィールドの割り付け
#pragma pmodule pm 指定	コア番号指定（マルチコア用）

(1) データとプログラムのセクション割り当て
C 言語レベルでモジュールやデータを任意のセクションに割り当てることができます。
なお，記述方法についての詳細は「4.2.4.1 データとプログラムのセクション割り当て」を参照してください。

```
#pragma section 属性指定文字 " セクション名 "
```

```
#pragma section 属性指定文字
```

```
#pragma section [ 文字列 ]
```

(2) アセンブラ命令の記述
C 言語中に，アセンブラ命令を記述することができます。
なお，記述方法についての詳細は「4.2.4.2 アセンブラ命令の記述」を参照してください。

```
#pragma inline_asm [(] 関数名 [(size= 数値 )][, 関数名 [(size= 数値 )], ...][)]
```

(3) インライン展開

(a) インライン展開指定
インライン展開する関数を指定することができます。
なお，インライン展開についての詳細は「4.2.4.3 インライン展開」を参照してください。

```
#pragma inline [(] 関数名 [, 関数名 , ...][)]
```

(b) インライン展開対象外関数指定
インライン展開を抑止する関数を指定することができます。
なお，インライン展開対象外関数指定についての詳細は「4.2.4.3 インライン展開」を参照してください。

```
#pragma noinline [(] 関数名 [, 関数名 , ...][)]
```

(4) 割り込み／例外ハンドラ指定
割り込み／例外処理ハンドラを C 言語で記述します。
なお，記述方法については「(3) 割り込み／例外ハンドラの記述方法」を参照してください。

```
#pragma interrupt [(] 関数名 [( 割り込み仕様 [, 割り込み仕様 ]...)] [,...] [)]
```

(5) 割り込み禁止関数指定
関数全体を割り込み禁止にします。
なお，記述方法についての詳細は「(2) 関数全体の割り込みを禁止する方法」を参照してください。

```
#pragma block_interrupt [(] 関数名 [)]
```

(6) 構造体パッキング指定
構造体パッキングを指定します。数値はパッキング値，すなわちメンバのアライメント値を指定します。数値には 1，2，4 が指定できます。数値を指定しない場合，デフォルトのアライメントとなります。
なお，記述方法についての詳細は「4.2.4.8 構造体パッキング」を参照してください。

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

(7) ビット・フィールドの割り付け
ビット・フィールドの並び順の切り替えを指定します。

なお，記述方法についての詳細は「4.2.4.9 ビット・フィールドの割り付け」を参照してください。

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

(8) コア番号指定（マルチコア用）
コア番号指定を行う関数を指定することができます。
なお，コア番号指定についての詳細は「4.2.4.10 コア番号指定（マルチコア用）」を参照してください。

```
#pragma pmodule pm 指定
```

4.2.4 拡張言語仕様の使用方法

この項では，下記の拡張機能の使用方法について説明します。

- データとプログラムのセクション割り当て
- アセンブラ命令の記述
- インライン展開
- 割り込みレベルの制御
- 割り込み／例外処理ハンドラ
- マスカブル割り込みの禁止／許可
- 組み込み関数
- 構造体パッキング
- ビット・フィールドの割り付け
- コア番号指定（マルチコア用）

4.2.4.1 データとプログラムのセクション割り当て

CC-RH では，C 言語レベルでモジュールやデータを任意のセクションに割り当てることができます。これにより，それぞれのセクション再配置属性の性質を利用した配置を指定することができます。

(1) #pragma section 指令
#pragma section 指令で次のような書式で記述します。

```
#pragma section 属性指定文字 " セクション名 "
変数の宣言／定義
```

```
#pragma section 属性指定文字
変数宣言／定義
```

```
#pragma section [ 文字列 ]
変数宣言／定義
```

属性指定文字でセクション再配置属性を，“ セクション名 ” でセクション名をそれぞれ指定します。本 pragma 以降に記述された静的変数はセクション再配置属性に沿って，指定されたセクションに配置されます。

“#pragma section 属性指定文字 ” の形式と “#pragma section [文字列]” の形式で，「属性指定文字」と「文字列」が同一の場合は，メッセージを出さずに “#pragma section 属性指定文字 ” の形式とみなします。ただし，-Xcheck=shc オプションを指定した場合は，“#pragma section 属性指定文字 ” の形式とみなした場合にメッセージが出力されます。
「属性指定文字」と「文字列」の同一性は，大小文字を含めて同一性を確認します。そのため，たとえば，大文字の R0_DISP32 は属性指定文字とみなしません。

例

```
#pragma section r0_disp32        /* これは data が属性指定文字と一致するので，*/
                                 /*“#pragma section 属性指定文字 ” の形式とする */
#pragma section R0_DISP32        /* これは属性指定文字と大小文字が異なるので，*/
                                 /*“#pragma section [ 文字列 ]” の形式とする */
```

セクション名が数値（0 ～ 9）で始まる場合，セクション名先頭に “_” を付加します。

例　　#pragma section 123 → セクション名：_123.bss，_123.const，_123.text，...

文字列に使用可能な文字は，次のとおりとする。
- 0 ～ 9
- a ～ z，A ～ Z
- _
- @
- .

#pragma section 系では，「宣言，または定義のいずれか一方のみ #pragma section が指定されている」場合，#pragma section 指定されている側が有効となります。
また，「宣言，または定義で別の #pragma section が指定されている」場合，最初に現れた #pragma section 指定が有効となります。

(a) “#pragma section 属性指定文字 " セクション名 " ” の指定方法
デフォルト・セクション名を変更し，#pragma 以降で宣言した変数を新しいセクション名で配置します。

- 本 pragma 指令は，デフォルト・セクション名をユーザ指定のセクション名に変更します。
- 本 pragma 指令は，外部変数／文字列リテラル／関数内 static 変数／関数に対して有効です。
- 本 pragma 指令は，記述された時点で以前の同じ属性への指定は無効になり，新たなセクション指定が有効となります。新たなセクションを指定しない場合，ファイルの終わりまで有効です。
- 再配置属性に default を指定した場合，それ以降 #pragma section のないデフォルトの割り付けに戻ります。default はすべての #pragma section 記述が無効になります。
- 変数の初期化がない場合は，指定された再配置属性に対し対応する bss 系のセクションに配置されます。
- 文字列リテラルは，デフォルトでは .const に割り付けられます。#pragma section に対しては他の const オブジェクトと同様に配置先が決定されます。
- セクション名には，ユーザ指定文字列に続けて，自動で再配置属性を示す文字列が付加されます。これは，初期化の有無により，自動で再配置属性を切り替えた場合，同名のセクションができてしまうのを避けるためです。
  例）#pragma section r0_disp32 "xxx" → セクション名：xxx.data
- const 再配置属性を指定した場合，const 修飾された変数のみセクション割り付けされ，修飾がない変数のセクションはデフォルトとなります。

表 4.16　　セクション再配置属性

対象	名前変更可能な属性指定文字	デフォルト・セクション名	ユーザ変更セクション名注
定義系 （data 系）	r0_disp16 r0_disp23 r0_disp32 ep_disp4 ep_disp5 ep_disp7 ep_disp8 ep_disp16 ep_disp23 gp_disp16 gp_disp23	.zdata .zdata23 .data .tdata4 .tdata5 .tdata7 .tdata8 .edata .edata23 .sdata .sdata23	*xxx*.zdata *xxx*.zdata23 *xxx*.data *xxx*.tdata4 *xxx*.tdata5 *xxx*.tdata7 *xxx*.tdata8 *xxx*.edata *xxx*.edata23 *xxx*.sdata *xxx*.sdata23

対象	名前変更可能な属性指定文字	デフォルト・セクション名	ユーザ変更セクション名注
宣言系 （bss 系）	初期値を持たない変数定義は bss 系としてコンパイラが自動判断します。そのため，bss 系用の属性指定文字はありません。	.zbss .zbss23 .bss .tbss4 .tbss5 .tbss7 .tbss8 .ebss .ebss23 .sbss .sbss23	*xxx*.zbss *xxx*.zbss23 *xxx*.bss *xxx*.tbss4 *xxx*.tbss5 *xxx*.tbss7 *xxx*.tbss8 *xxx*.ebss *xxx*.ebss23 *xxx*.sbss *xxx*.sbss23
const 系	const zconst zconst23	.const .zconst .zconst23	*xxx*.const *xxx*.zconst *xxx*.zconst23
text 系	text	.text	*xxx*.text
指定解除	default	すべての指定を解除して，デフォルトのセクション名に戻します。	

注 セクション名を “*xxx*” と書いた場合の実セクション名です。

例 1.

```
#pragma section r0_disp32   "mydata"
int i = 0;                      /*“mydata.data” に割り当てられる */
int j;                          /*“mydata.bss” に割り当てられる */

#pragma section gp_disp16   "mydata2"
int i2 = 1;                     /*“mydata2.sdata” に割り当てられる */

#pragma section const       "myconst"
const int c2 = 0;               /*“myconst.const” に割り当てられる */
int j2;                         /*“.bss” に割り当てられる */
```

例 2.

```
#pragma section ep_disp4    "XXX"
extern  int     i;
void func()
{
        i = 5;
}
#pragma section r0_disp32 "OOO"
int     i = 5;                  /*"XXX.tdata4" に割り当てられる */
                                /* 最初の宣言にかかる #pragma section がある場合，*/
                                /* それが有効 */
```

例 3.

```
#pragma section ep_disp4
int     i;
#pragma section ep_disp23
int     i;                      /*".tbss4" に割り当てられる */
                                /* 最初の宣言にかかる #pragma section がある場合，*/
                                /* それが有効 */
```

例 4.

```
<def.c>
#pragma section ep_disp4
extern  int     i;
#pragma section ep_disp4 "XXX"
int i = 5;                      /*".tdata4" に割り当てられる */
                                /* 最初の宣言にかかる #pragma section がある場合，*/
                                /* それが有効 */

<use.c>
#pragma section ep_disp4
extern  int     i;
void func()
{
        i = 5;                  /* アクセスに問題なし */
}
```

例 5.

```
#pragma section ep_disp4    "XXX"
extern  int     i;
#pragma section ep_disp4    "OOO"
int     i = 5;                  /*".tdata4" に割り当てられる */
                                /* 最初の宣言にかかる #pragma section がある場合，*/
                                /* それが有効 */
```

例 6.

```
#pragma section r0_disp16   "myzdata"
#pragma section const       "myconst"   /* この行で "myzdata" の指定はなくなる */
int     i = 0;                          /*.data const でないので効かない */
const int       ci = 0;                 /*myconst.const*/
#pragma section r0_disp16   "myzadata"  /* 再度指定 */
int     j = 0;                          /*myzdata.zdata になる */
const int       cj = 0;                 /*.const 直前の pragma で myconst 指定は */
                                        /* 打ち消され ".const" に配置される */
```

例 7.

```
extern  int     i;

#pragma section ep_disp4
int     i;      /*".tdata4" に配置する */
                /* 宣言側と相違あるが，仮に他翻訳単位で */
                /*#pragma section ep_disp23 なしでアクセスしたとしても，*/
                /* 長距離（効率悪い）アドレッシング・モードでアクセスが正常動作する */
```

例 8.

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

#pragma section ep_disp4
int     i;      /*".tdata4" に配置する */
                /* デフォルト・セクション名に戻す #pragma section default は，*/
                /* セクション指定とみなさない */
                /*extern 宣言側は，セクション指定がないものとみなす */
```

(b) “#pragma section 属性指定文字 ” の指定方法
#pragma 以降で宣言した変数をデフォルト・セクションに配置します。

- デフォルトのセクション名で，データの配置位置を指定します。

- デフォルトのセクション名に配置されることを除けば，配置規則は “#pragma section 属性指定文字 " セクション名 " ” と同一です。

表 4.17 セクション再配置属性

対象	属性指定文字	デフォルト・セクション名
定義系 （data 系）	r0_disp16 r0_disp23 r0_disp32 ep_auto ep_disp4 ep_disp5 ep_disp7 ep_disp8 ep_disp16 ep_disp23 gp_disp16 gp_disp23	.zdata .zdata23 .data .tdata4/.tdata5/.tdata7/.tdata8 から自動選択 .tdata4 .tdata5 .tdata7 .tdata8 .edata .edata23 .sdata .sdata23
宣言系 （bss 系）	初期値を持たない変数定義は bss 系としてコンパイラが自動判断します。そのため，bss 系用の属性指定文字はありません。	.zbss .zbss23 .bss .tbss4 .tbss5 .tbss7 .tbss8 .ebss .ebss23 .sbss .sbss23
const 系	const zconst zconst23	.const .zconst .zconst23
text 系	text	.text
指定解除	default	すべての指定を解除して，デフォルトのセクション名に戻します。

例

```
#pragma section gp_disp16
int a = 1;                          /*.sdata セクションに配置する */
int b;                              /*.sbss セクションに配置する */
```

- ep_auto は宣言の型に応じて，ep_disp4/ep_disp5/ep_disp7/ep_disp8 のいずれかとして動作します。

宣言の型	対応する ep_disp*n*
unsigned char	ep_disp4
unsigned short	ep_disp5
signed char char _Bool	ep_disp7

宣言の型	対応する ep_disp*n*
signed short signed int unsigned int signed long unsigned long signed long long unsigned long long float double long double ポインタ 列挙型（int 型サイズの場合）	ep_disp8

備考 1. 配列型は，要素の型に応じて対応する ep_disp*n* を選択します。ただし，多次元配列（配列の配列）の場合は，多次元配列に格納する要素の型とします。

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

備考 2. -Xenum_type オプションの作用により列挙型が int よりも小さな型になる場合，小さくなった型に応じて対応する ep_disp*n* を選択します。

備考 3. 構造体および共用体は，.tdata4/.tdata5/.tdata7/.tdata8 のいずれにも配置しません。

- ep_auto は宣言の型に応じて，ep_disp4/ep_disp5/ep_disp7/ep_disp8 のいずれかとして動作するため，宣言と定義で ep_auto と ep_disp*n* が同一の意味になる場合はエラーになりません。異なる意味となる場合はコンパイル・エラーとなります。

例 1.

```
#pragma section ep_auto
extern  unsigned char   ch;
#pragma section ep_disp4
unsigned char           ch;     /* エラーも警告もなしで，.tdata4 に配置する */
```

例 2.

```
#pragma section ep_auto
extern  unsigned char   ch;
#pragma section ep_disp7
unsigned char           ch;     /* エラーも警告もなしで，.tdata4 に配置する */
                                /* 最初の宣言にかかる #pragma section がある場合，*/
                                /* それが有効 */
```

(c) “#pragma section [文字列]” の指定方法
コンパイラが出力するセクション名を切り替えます。
デフォルトのセクション名と切り替え後のセクション名は，次のとおりです。

対象	指定方法	デフォルト	切り替え後
プログラム	#pragma section *xxx*	text	*xxx*.text
定数		const	*xxx*.const
初期値ありデータ		data	*xxx*.data
初期値なしデータ		bss	*xxx*.bss

- 文字列を省略した場合，すべてのセクション名をデフォルトに設定します。

例

```
#pragma section abc
int a;                  /*a はセクション abc.bss に割り付きます */
const int d=1;          /*d はセクション abc.const に割り付きます */
void f(void)            /*f はセクション abc.text に割り付きます */
{
        a = d;
}

#pragma section
int b;                  /*b はセクション .bss に割り付きます */
void g(void)            /*g はセクション .text に割り付きます */
{
        b = d;
}

#pragma section 123
int c;                  /*c はセクション _123.bss に割り付きます */
void h(void)            /*h はセクション _123.text に割り付きます */
{
        c = d;
}
```

4.2.4.2 アセンブラ命令の記述

CC-RH では，C ソース・プログラム中にアセンブラ命令が記述できます。

(1) #pragma 指令
アセンブラ命令を埋め込む #pragma 指令には，#pragma inline_asm があります。
これは関数そのものをアセンブラ命令のみとみなして，呼び出し箇所にインライン展開します。

```
#pragma inline_asm [(] 関数名 [(size= 数値 )][, 関数名 [(size= 数値 )], ...][)]
```

#pragma inline_asm で宣言したアセンブリ記述関数をインライン展開します。
アセンブラ埋め込みインライン関数の呼び出し規則は通常関数の呼び出し規則と同様です。
（size= 数値）を指定しても，コンパイル結果には影響しません。

例

- C ソース

```
#pragma inline_asm      func_add
static int      func_add(int a, int b){
        add     r6, r7
        mov     r7, r10
}
void func(int *p){
        *p = func(10,20);
}
```

- 出力コード

```
_func:
prepare r20, 0
mov     r6, r20
movea   0x0014, r0, r7
mov     10, r6
add     r6, r7
mov     r7, r10
```

(2) #pragma inline_asm 使用時の注意事項

- #pragma inline_asm は，関数本体の定義の前に指定してください。
- #pragma inline_asm で指定した関数に対しても外部定義を生成します。
- アセンブラ埋め込みインライン関数内で関数の出入口で保証するレジスタを使用する場合は，アセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタの退避／復帰が必要です。
- コンパイラは #pragma inline_asm 内に書いた文字列を，チェックも変更もなしで，そのままアセンブラに渡します。
- （size= 数値）で指定する数値は，正の整数定数のみです。数値が浮動小数点数，または 0 以下の数値の場合，エラーとなります。
- static 関数に #pragma inline_asm を指定した場合，関数定義はインライン展開後に削除されます。
- アセンブリ記述は，プリプロセッサの処理対象となります。このため，アセンブリ言語で使用される命令やレジスタと同じ名前のマクロ（例："MOV" や "r5" など）を #define でマクロ定義する場合は注意してください。
- RH850 のアセンブリ言語は，# で始まるコメントを使用できますが，この # コメントを使うと，プリプロセッサは前処理指令と解釈するため，アセンブリ記述関数内では # コメントを使わないでください。
- アセンブリ記述関数内にラベルを書くと，インライン展開の数だけ同一名のラベルが作られてしまいます。この場合は，アセンブリ記述のローカル・ラベルを使ってください。ローカル・ラベルはアセンブリ・ソースでは同一名ですが，アセンブラが自動的に別名に変換します。

4.2.4.3 インライン展開

CC-RH では，関数ごとのインライン展開ができます。ここでは，インライン展開の指定について説明します。

(1) インライン展開とは

インライン展開とは，関数呼び出し部分に関数本体を展開することを言います。これにより，関数呼び出しによるオーバーヘッドが小さくなり，また，最適化の可能性が高められることから，実行速度向上を図ることができます。
ただし，インライン展開を行うと，オブジェクト・サイズは増大することになります。
インライン展開したい関数は，#pragma inline で指定します。

```
#pragma inline [(] 関数名 [, 関数名 , ...][)]
```

関数名は，C 言語記述の関数名を記述してください。たとえば，"void func1 ( ) { }" という関数であれば "func1" と指定します。また，関数名は “,”（カンマ）で区切って複数指定することができます。

```
#pragma inline  func1, func2
void    func1() {...}
void    func2() {...}
void func(void) {
    func1();    /* インライン展開対象 */
    func2();    /* インライン展開対象 */
}
```

(2) インライン展開の条件

#pragma inline 指定された関数をインライン展開するためには，最低限次の条件が必要となります。
ただし，CC-RH の内部処理の関係により，次の条件を満たしていてもインライン展開されない場合があります。

(a) インライン展開を “ する関数 ” と “ される関数 ” を同一ファイル内に記述する

インライン展開を “ する関数 ” と “ される関数 ”，つまり，“ 関数呼び出し ” と “ 関数定義 ” は “ 同一ファイル内 ” に存在しなければなりません。別の C ソースに書かれてある関数をインライン展開することはできません。
この場合，CC-RH はエラーも警告メッセージも出力せず，インライン展開指定を無視します。

(b) #pragma inline を “ 関数定義より前 ” に記述する

pragma inline が，関数定義よりも後ろに記述されていた場合，警告を出力してインライン展開指定を無視します。ただし，関数のプロトタイプ宣言との記述順序は問いません。次に例を示します。

例

【インライン展開指定：有効】	【インライン展開指定：無効】

`#pragma inline func1, func2`
`void func1(); /* プロトタイプ宣言 */`
`void func2(); /* プロトタイプ宣言 */`
`void func1() {...} /* 関数定義 */`
`void func2() {...} /* 関数定義 */`
	`void func1(); /* プロトタイプ宣言 */`
	`void func2(); /* プロトタイプ宣言 */`
	`void func1() {...} /* 関数定義 */`
	`void func1() {...} /* 関数定義 */`
	`#pragma inline func1, func2`

(c) インライン展開する関数の“呼び出し”と“定義”の間で，“引数の数”を同じにする
インライン展開する関数の“呼び出し”と“定義”の間で“引数の数”が違う場合，インライン展開指定を無視します。

(d) インライン展開する関数の“呼び出し”と“定義”の間で，“戻り値の型”や“引数の型”を同じにする
インライン展開する関数の“呼び出し”と“定義”の間で，“戻り値の型”や“引数の型”が異なる場合，インライン展開指定を無視します。ただし，引数の型が整数型（enum を含む），またはポインタ型でサイズが同じ場合は，インライン展開を行います。

(e) インライン展開する関数の引数は“可変個”にしない
引数が“可変個”の関数にインライン展開指定した場合，エラーも警告メッセージも出力せず，インライン展開指定を無視します。

(f) “再帰関数”はインライン展開できない
自分自身を呼び出す“再帰関数”をインライン展開指定した場合，エラーも警告メッセージも出力せず，インライン展開指定を無視します。ただし，関数呼び出しが複数ネストし，そのネストした中に自分自身を呼び出すコードが存在した場合，インライン展開する場合があります。

(g) 関数内で仮引数のアドレスを参照しない
関数内で仮引数のアドレスを参照した場合，エラーも警告メッセージも出力せず，インライン展開指定を無視します。

(h) 展開対象関数のアドレスを介して呼び出しを行わない
展開対象関数のアドレスを介して呼び出しを行った場合，エラーも警告メッセージも出力せず，インライン展開指定を無視します。

(i) “割り込みハンドラ”はインライン展開できない
#pragma interrupt で記述された関数は“割り込みハンドラ”として認識されますが，この関数に対してインライン展開指定した場合，警告メッセージを出力して，インライン展開指定を無視します。

(j) #pragma block_interrupt 指定で関数内を割り込み禁止にすると，インライン展開できない
#pragma block_interrupt で，関数内を割り込み禁止として宣言された関数に対してインライン展開指定した場合，警告メッセージを出力して，インライン展開指定を無視します。

(k) #pragma inline_asm 指定をした場合は #pragma inline 指定を無視する
#pragma inline_asm で，アセンブリ記述関数をインライン展開指定した場合，警告メッセージを出力して，#pragma inline 指定を無視します。

(l) -Xmerge_files を指定した場合は，同一ファイル内に記述していなくても展開される場合がある

(3) インライン展開対象外関数
-Oinline オプション使用時に特定関数のインライン展開を抑止したい場合，インライン展開を抑止したい関数は，#pragma noinline で指定します。

```
#pragma noinline [(] 関数名 [, 関数名 , ...][)]
```

同一翻訳単位内で，同じ関数に対して #pragma inline と #pragma noinline を同時に指定するとエラーとなります。

(4) オプション指定によるインライン展開動作の違いの例
#pragma inline 指定とオプション指定による“インライン展開動作の違い”は，次のようになります。

-Oinline=0	エラーも警告メッセージも出力せず，インライン展開指定を無視します。
-Oinline=1	インライン展開指定した関数をインライン展開します。
-Oinline=2	インライン展開指定していない関数でも自動的にインライン展開します。 ただし，インライン展開対象外関数指定した関数は，インライン展開しません。
-Oinline=3	インライン展開指定していない関数でも自動的にインライン展開します。 ただし，インライン展開対象外関数指定した関数は，インライン展開しません。

(5) インライン展開の例
インライン展開の例は，次のようになります。

- C ソース

```
#pragma inline  (func)
static int      func(int a, int b)
{
        return  (a+b)/2;
}
int     x;
main()
{
        x = func (10, 20);
}
```

- 展開イメージ

```
int     x;
main()
{
        int     func_result;
{
        int     a_1 = 10, b_1 = 20;
        func_result = (a_1+b_1)/2;
}
        x = func_result;
}
```

4.2.4.4 割り込みレベルの制御

CC-RH では，RH850 ファミリの割り込みに対して，C ソース上で，次の制御を行うことができます。

- 割り込み優先順位レベルの制御
- マスカブル割り込みの受け付けの許可／禁止（割り込みのマスク）

つまり，“ 割り込み制御レジスタ ” を操作することができます。

(1) 割り込み優先順位レベルの制御
“ 割り込み優先順位レベル ” を制御する場合は，“__set_il 関数 ” を用いて次のように指定します。

```
__set_il_rh(int 割り込みの優先順位レベル , void* 割り込み制御レジスタのアドレス );
```

“ 割り込みの優先順位レベル ” として指定できる値は “1 ～ 16” の整数値です。RH850 の割り込み優先順位レベルは “0 ～ 15 までの 16 段階 ” を指定するため，“RH850 の割り込みの優先順位レベルを 5 にしたい ” 場合は，この関数で指定する割り込みの優先順位レベルは “6” と指定します。

(2) マスカブル割り込みの受け付けの許可／禁止
“ 割り込みに対して，マスカブル割り込みの受け付けの許可／禁止 ” を制御する場合は，次のように指定します。

```
__set_il_rh(int マスカブル割り込みの許可／禁止 , void* 割り込み制御レジスタのアドレス );
```

“ マスカブル割り込みの許可／禁止 ” に設定できる値は “-3 ～ 0” の整数値です。

設定値	動作
0	マスカブル割り込みの受け付け許可（割り込みのマスクを解除）
-1	マスカブル割り込みの受け付け禁止（割り込みをマスク）
-2	割り込みベクタ方式を直接分岐方式 (標準仕様) にしたい場合
-3	割り込みベクタ方式をテーブル参照方式 (拡張仕様) にしたい場合

4.2.4.5 割り込み／例外処理ハンドラ

CC-RH では，C 言語で “ 割り込み ” や “ 例外 ” が発生したときに呼ばれる “ 割り込みハンドラ ”，“ 例外ハンドラ ” を記述することができます。ここでは，その記述方法などについて説明します。

(1) 割り込み／例外の発生
RH850 ファミリでは，割り込みや例外が発生すると，その割り込みや例外に対応したハンドラ・アドレスにジャンプします。
なお，ハンドラ・アドレスの並びや，搭載している割り込みは，RH850 の品種ごとに異なります。詳細は，使用する各デバイスのユーザーズ・マニュアルを参照してください。
具体的な記述方法は「(3) 割り込み／例外ハンドラの記述方法」で説明します。

(2) 割り込み／例外発生時に行う必要のある処理
関数実行時に割り込み／例外が入ると，即座に割り込み／例外処理を行う必要があります。そして割り込み／例外処理が終わると，割り込みが入った時点の関数に戻る必要があります。
したがって，割り込み／例外発生時には，そのときのレジスタ情報を保存し，割り込み／例外処理が終わった後は，そのレジスタ情報を復帰する必要があります。

(3) 割り込み／例外ハンドラの記述方法
割り込み／例外ハンドラの記述上の形態は，通常の C 言語関数と変わりませんが，C 言語で記述した関数を，CC-RH に対して “ 割り込み／例外ハンドラ ” として認識させる必要があります。CC-RH では，割り込み／例外ハンドラの指定を “#pragma interrupt 指令 ” で行います。

```
#pragma interrupt   [(] 関数名 [( 割り込み仕様 [, 割り込み仕様 ]...)] [,...] [)]
```

関数名は，C 言語記述の関数名を記述してください。たとえば，"void func1 ( ) { }" という関数であれば "func1" と指定します。
割り込み関数の戻り型は，常に void としてください。
割り込み関数は，関数の出口コードが通常関数と異なるため，通常関数のように呼び出さないでください。

(a) 割り込み関数定義
割り込み関数定義は，引数なし，または引数 1 個のみ使用できます。

例 1. 引数なしの場合

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

例 2. 引数 1 個の場合

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

(b) 割り込み仕様
割り込み仕様には，以下のものを指定できます。

enable=	多重割込の可否を指定します。true/false/manua が記載できます。 - true ei/di を出力します。 eipc/eipsw の退避／復帰コードを出力します。 - false（デフォルト） ei/di を出力しません。 eipc/eipsw の退避／復帰コードを出力しません。 - manual ei/di を出力しません。 eipc/eipsw の退避／復帰コードを出力します。

priority= channel=	priority，または channel のどちらか一方のみ記述可能です（両方書いた場合は，コンパイル・エラーとなります）。 - priority= 例外発生要因を指定します。以下の字句から 1 個のみを書くことができます。 SYSERR/HVTRAP/FETRAP/TRAP0/TRAP1/RIE/FPP 注 1/FPI 注 1/FPINT 注 2/UCPOP/MIP/MDP/ITLBE/DTLBE/PIE/MAE/FENMI/FEINT/EINT_PRIORITYX（X は 0 から 15） - channel= 割込発生チャネルを指定します。割り込みの「拡張仕様」を使う場合に選択してください。 EI ハンドラと判断してコードを生成します。 priority，または channel を記載しなかった場合は，EIINT と判断します。
fpu=	fpu のコンテキスト fpepc/fpsr を退避／復帰するか指定します。true/false/auto が記載できます。 - true fpepc/fpsr を退避／復帰します。 - false fpepc/fpsr を退避／復帰しません。 - auto（デフォルト） -Xfloat オプションが指定されている場合は，true を指定したとみなします。 -Xfloat=soft の場合は，false を指定したとみなします。
callt=	callt の文脈 ctpc/ctpsw を退避／復帰するか指定します。true/false が記載できます。 - true（デフォルト） ctpc/ctpsw を退避／復帰します。 - false ctpc/ctpsw を退避／復帰しません。
registry=	最適化リンカが割込テーブルを生成し，生成した割込テーブルに当該例外ハンドラを自動登録するか否かを指定します。auto/manual が記載できます。 - auto priproty，または channel の記述があればテーブルの生成とテーブルへの登録を最適化リンカが実施します。 - priority 指定時 割込チャネルごとの割込テーブルを作成し，作成したテーブルの，オプション priority で指定したエントリに当該例外ハンドラを登録します。 さらに，作成した割込テーブルのアドレスをレジスタ RBASE/EBASE に代入する処理をスタートアップコードに挿入します。 - channel 指定時 割込チャネルごとの割込テーブルを作成し，作成したテーブルの，オプション channel で指定したエントリに当該例外ハンドラを登録します。 さらに，作成した割込テーブルのアドレスをレジスタ INTBP に代入する処理をスタートアップコードに挿入します。 - manual（デフォルト） 割込テーブルの生成も割込テーブルへの登録も行いません。

注 1. -Xcpu=g3mh 指定時は使用できません。使用した場合はエラーになります。

注 2. -Xcpu=g3mh 指定時のみ使用できます。それ以外で使用した場合はエラーになります。

割り込み仕様を書く場合は，必ず = の右項を必要とします。
たとえば，“enable=” だけを書くとコンパイル・エラーとなります。割り込み仕様のデフォルトとは，個々の割り込み仕様を書かない場合の動作を意味します。

(c) EI レベル例外の出力コード内容
EI レベル例外の割り込み関数に対して，コンパイラが入口／出口に挿入する命令列を以下に示します。主に，EIINT や FPI 等がこれに該当します。
ただし，これらをすべての割り込み関数に挿入するわけではなく，ユーザの #pragma 記述やコンパイル・オプション等に応じて必要な処理が出力されます。

<1> [割り込み関数の入口コード]
（1）コンテキストの退避に使用するスタック領域を確保
（2）割り込み関数中で使用する関数呼び出し前後で内容が保証されないレジスタを退避
（3）EIPC，EIPSW を退避
（4）仮引数を記述した関数の場合，EIIC を R6 に設定
（5）多重割り込みを許可
（6）CTPC，CTPSW を退避
（7）FPEPC，FPSR を退避

<2> [割り込み関数の出口コード]
（8）インプレサイス割り込み待ちを設定
（9）FPEPC，FPSR を復帰
（10）CTPC，CTPSW を復帰
（11）多重割り込みを禁止
（12）EIPC，EIPSW を復帰
（13）割り込み関数中で使用した関数呼び出し前後で内容が保証されないレジスタを復帰
（14）コンテキストの退避に使用したスタック領域を解放
（15）eiret

次に，具体的な出力コード例を示します。コード例中の番号（1）～（15）は，上記の各処理に記載された番号と対応しています。
なお，必ず出力コード例のとおりの命令が出力されるわけではありません。使用する命令や汎用レジスタ等は，コード例とは異なる場合があります。

例 1. EI レベル例外の出力例 1

```
#pragma interrupt func1(enable=true, callt=true, fpu=true)
void func1(unsigned long eiic)
{
        ユーザ記述の処理；
}
```

```
_func1:
    movea    -0x00000038, r3, r3      ; (1)
    st23.dw r6, 0x00000030[r3]        ; (2)
    stsr     0, r6                    ; (3)
    stsr     1, r7                    ; (3)
    st23.dw r6, 0x00000028[r3]        ; (3)
    stsr     13, r6                   ; (4)
    ei                                ; (5)
    st23.dw r4, 0x00000020[r3]        ; (2)
    st23.dw r8, 0x00000018[r3]        ; (2)
    st23.dw r10, 0x00000010[r3]       ; (2)
    stsr     16, r8                   ; (6)
    stsr     17, r9                   ; (6)
    st23.dw r8, 0x00000008[r3]        ; (6)
    stsr     7, r8                    ; (7)
    stsr     6, r9                    ; (7)
    st23.dw r8, 0x00000000[r3]        ; (7)
    prepare ....                      ; Callee-Save レジスタ退避
        :                             ; ユーザが記述した処理
    dispose ....                      ; Callee-Save レジスタ復帰
    synce                             ; (8)
    ld23.dw 0x00000000[r3], r8        ; (9)
    ldsr     r9, 6                    ; (9)
    ldsr     r8, 7                    ; (9)
    ld23.dw 0x00000008[r3], r8        ; (10)
    ldsr     r9, 17                   ; (10)
    ldsr     r8, 16                   ; (10)
    ld23.dw 0x00000010[r3], r10       ; (13)
    ld23.dw 0x00000018[r3], r8        ; (13)
    ld23.dw 0x00000020[r3], r4        ; (13)
    di                                ; (11)
    ld23.dw 0x00000028[r3], r6        ; (12)
    ldsr     r7, 1                    ; (12)
    ldsr     r6, 0                    ; (12)
    ld23.dw 0x00000030[r3], r6        ; (13)
    movea    0x00000038, r3, r3       ; (14)
    eiret                             ; (15)
```

割り込み関数先頭にコンパイラが埋め込む入口コード

ユーザが記述した割り込み処理

割り込み関数末尾にコンパイラが埋め込む出口コー

例 2. EI レベル例外の出力例 2
仮引数なし，割り込みの多重化は手動（enable=manual）の場合

```
#pragma interrupt func1(enable=true, callt=true, fpu=true)
void func1(unsigned long eiic)
 {
        ユーザ記述の処理；
}
```

```
_func1:
    movea   -0x00000038, r3, r3     ; (1)
    st23.dw r6, 0x00000030[r3]      ; (2)
    stsr    0, r6                   ; (3)
    stsr    1, r7                   ; (3)
    st23.dw r6, 0x00000028[r3]      ; (3)
    st23.dw r4, 0x00000020[r3]      ; (2)
    st23.dw r8, 0x00000018[r3]      ; (2)
    st23.dw r10, 0x00000010[r3]     ; (2)
    stsr    16, r8                  ; (6)
    stsr    17, r9                  ; (6)
    st23.dw r8, 0x00000008[r3]      ; (6)
    stsr    7, r8                   ; (7)
    stsr    6, r9                   ; (7)
    st23.dw r8, 0x00000000[r3]      ; (7)
    prepare ....                    ; Callee-Save レジスタ退避
        :                           ; ユーザが記述した処理
    dispose ....                    ; Callee-Save レジスタ復帰
    ld23.dw 0x00000000[r3], r8      ; (9)
    ldsr    r9, 6                   ; (9)
    ldsr    r8, 7                   ; (9)
    ld23.dw 0x00000008[r3], r8      ; (10)
    ldsr    r9, 17                  ; (10)
    ldsr    r8, 16                  ; (10)
    ld23.dw 0x00000010[r3], r10     ; (13)
    ld23.dw 0x00000018[r3], r8      ; (13)
    ld23.dw 0x00000020[r3], r4      ; (13)
    ld23.dw 0x00000028[r3], r6      ; (12)
    ldsr    r7, 1                   ; (12)
    ldsr    r6, 0                   ; (12)
    ld23.dw 0x00000030[r3], r6      ; (13)
    movea   0x00000038, r3, r3      ; (14)
    eiret                           ; (15)
```

割り込み関数先頭にコンパイラが埋め込む入口コード

ユーザが記述した割り込み処理

割り込み関数末尾にコンパイラが埋め込む出口コード

(d) FE レベル例外の出力コード内容

FE レベル例外の割り込み関数に対して，コンパイラが入口／出口に挿入する命令列を以下に示します。主に，FEINT や PIE 等がこれに該当します。

ただし，これらをすべての割り込み関数に挿入するわけではなく，ユーザの #pragma 記述やコンパイル・オプション等に応じて必要な処理が出力されます。

<1> [割り込み関数の入口コード]

（1）コンテキストの退避に使用するスタック領域を確保
（2）割り込み関数中で使用する関数呼び出し前後で内容が保証されないレジスタすべてを退避
（3）仮引数を記述した関数の場合，FEIC を R6 に設定
（4）CTPC，CTPSW を退避
（5）FPEPC，FPSR を退避

<2> [割り込み関数の出口コード]

（6）FPEPC，FPSR を復帰
（7）CTPC，CTPSW を復帰
（8）割り込み関数中で使用する関数呼び出し前後で内容が保証されないレジスタすべてを復帰
（9）コンテキストの退避に使用したスタック領域を解放
（10）feret

次に，具体的な出力コード例を示します。コード例中の番号（1）～（10）は，上記の各処理に記載された番号と対応しています。

なお，必ず出力コード例のとおりの命令が出力されるわけではありません。使用する命令や汎用レジスタ等は，コード例とは異なる場合があります。

例　　FE レベル例外の出力例

```
#pragma interrupt func1(priority=feint, callt=true, fpu=true)
void func1(unsigned long feic)
{
        ユーザ記述の処理；
}
```

```
_func1:
    movea    -0x00000030, r3, r3      ; (1)
    st23.dw r4, 0x00000028[r3]        ; (2)
    st23.dw r6, 0x00000020[r3]        ; (2)
    st23.dw r8, 0x00000018[r3]        ; (2)
    st23.dw r10, 0x00000010[r3]       ; (2)
    stsr     14, r6                   ; (3)
    stsr     16, r8                   ; (4)
    stsr     17, r9                   ; (4)
    st23.dw r8, 0x00000008[r3]        ; (4)
    stsr     7, r8                    ; (5)
    stsr     6, r9                    ; (5)
    st23.dw r8, 0x00000000[r3]        ; (5)
    prepare ....                      ; Callee-Save レジスタ退避
        :                             ; ユーザが記述した処理
    dispose ....                      ; Callee-Save レジスタ復帰
    ld23.dw 0x00000000[r3], r8        ; (6)
    ldsr     r9, 6                    ; (6)
    ldsr     r8, 7                    ; (6)
    ld23.dw 0x00000008[r3], r8        ; (7)
    ldsr     r9, 17                   ; (7)
    ldsr     r8, 16                   ; (7)
    ld23.dw 0x00000010[r3], r10       ; (8)
    ld23.dw 0x00000018[r3], r8        ; (8)
    ld23.dw 0x00000020[r3], r6        ; (8)
    ld23.dw 0x00000028[r3], r4        ; (8)
    movea    0x00000030, r3, r3       ; (9)
    feret                             ; (10)
```

FEINT 先頭にコンパイラが埋め込む入口コード

ユーザが記述した FEINT 処理

FEINT 末尾にコンパイラが埋め込む出口コード

(e) FE レベル例外（復帰／回復不可）の出力コード内容

FE レベル例外（復帰／回復不可）の割り込み関数に対して，コンパイラが入口／出口に挿入する命令列を以下に示します。主に，FENMI や SYSERR 等がこれに該当します。

<1> [割り込み関数の入口コード]

（1）仮引数を記述した関数の場合，FEIC を R6 に設定
仮引数を記述しない場合は，何も出力されません。

<2> [割り込み関数の出口コード]

なし

備考　コンテキストの退避／復帰は一切出力されません。
関数呼び出し前後で内容が保証されるレジスタの退避／復帰も出力されません。

次に，具体的な出力コード例を示します。コード例中の番号（1）は，上記の各処理に記載された番号と対応しています。

例　FE レベル例外（復帰／回復不可）の出力例

```
#pragma interrupt func1(priority=fenmi)
void func1(unsigned long feic)
{
        ユーザ記述の処理；
}
```

```
_func1:
    stsr    14, r6                  ; (1)

        :                           ; ユーザが記述した処理
        :
        :
```

FENMI 先頭に
コンパイラが
埋め込む入口コード

ユーザが記述した
FENMI 処理

末尾には，
出力されません

(4) 割り込み／例外ハンドラの記述時の注意事項

- 割り込み／例外ハンドラとして指定された関数はインライン展開できません。#pragma inline 指定しても無視されます。

4.2.4.6 マスカブル割り込みの禁止／許可

CC-RH では，C ソースにおいて，マスカブル割り込みを禁止にすることができます。
マスカブル割り込みを禁止する方法には，大きく分けて次の 2 通りがあります。

- 関数内で部分的に割り込みを禁止する方法
- 関数全体の割り込みを禁止する方法

(1) 関数内で部分的に割り込みを禁止する方法
C 言語で記述した関数内で，部分的に割り込みを禁止する場合，アセンブラ命令の “di 命令 ” と “ei 命令 ” を使用することができますが，CC-RH では C ソースで割り込み制御を行うことのできる関数を用意しています。

表 4.18 割り込み制御関数

割り込み制御関数	動作	CC-RH の処理
__DI	すべてのマスカブル割り込みの受け付けを禁止します。	di 命令を生成
__EI	すべてのマスカブル割り込みの受け付けを許可します。	ei 命令を生成

例 __DI，__EI 関数の記述方法とその出力コード

- C ソース

```
void func1(void) {
      :
    __DI();
    /* 割り込みを禁止して行いたい処理を記述 */
    __EI();
      :
}
```

- 出力コード

```
_func1:
    -- プロローグ・コード
      :
    di
    -- 割り込みを禁止して行いたい処理
    ei
      :
    -- エピローグ・コード
    jmp     [lp]
```

(2) 関数全体の割り込みを禁止する方法
CC-RH では，関数全体のマスカブル割り込み割り込みを禁止する “#pragma block_interrupt” 指令を用意しています。

次のような書式で記述します。

```
#pragma block_interrupt [(] 関数名 [)]
```

関数名は，C 言語記述の関数名を記述してください。たとえば，"void func1 ( ) { }" という関数であれば "func1" と指定します。
上記で " 関数名 " で指定された関数に対し，マスカブル割り込みを禁止します。「(1) 関数内で部分的に割り込みを禁止する方法」で説明したように，関数の最初に "__DI ( );" を，最後で "__EI ( );" を記述することもできますが，この場合だと，CC-RH が出力する " プロローグ・コード "，" エピローグ・コード " に対してマスカブル割り込みを禁止／許可することができず，関数全体を完全に割り込み禁止にすることができません。
#pragma block_interrupt 指令を用いると，" プロローグ・コード " 実行の直前にマスカブル割り込みが禁止され，" エピローグ・コード " 実行直後にマスカブル割り込み割り込みが許可されます。そのため関数全体を完全に割り込み禁止にすることができます。

例　#pragma block_interrupt 指令の使用方法と，出力されるコードは次のとおりです。

- C ソース

```
#pragma block_interrupt func1
void func1(void) {
      :
    /* 割り込みを禁止して行いたい処理を記述 */
      :
}
```

- 出力コード

```
_func1:
    di
    -- プロローグ・コード
      :
    -- 割り込みを禁止して行いたい処理
      :
    -- エピローグ・コード
    ei
    jmp     [lp]
```

(3) 関数全体の割り込み禁止時の注意事項
関数全体の割り込みを禁止にした場合の注意事項について，次に示します。

- 同じ関数に対して，割り込みハンドラ指定と #pragma block_interrupt 指定された場合，割り込みハンドラ指定の方が優先され，割り込み禁止の設定は無視されます。
- 割り込み禁止となっている関数内で，次の関数を呼び出した場合，その呼び出しからの復帰時に，割り込み許可状態になるため，注意が必要です。
- #pragma block_interrupt 指定された関数
- 関数の先頭で割り込み禁止をし，最後で割り込み許可している関数
- #pragma block_interrupt 指令は，関数の定義と同一ファイル内で，かつ，定義より前に記述してください。
- 関数の定義より後ろに記述された場合，コンパイル時にエラーとなります。
- ただし，関数のプロトタイプ宣言順とは関係ありません。
- #pragma block_interrupt 指定された関数は，#pragma inline 指令を指定したり，最適化オプションでインライン展開指定ができません。インライン展開指定は無視されます。
- #pragma block_interrupt 指定しても，PSW（プログラム・ステータス・ワード）内の EP フラグ（例外処理中を示すフラグ）の操作を行うコードは出力されません。

4.2.4.7 組み込み関数

CC-RH では，アセンブラ命令の一部を " 組み込み関数 " として C ソースに記述することができます。ただし，" アセンブラ命令そのもの " を記述するのではなく，CC-RH で用意した関数の形式で記述します。
組み込み関数の引数に暗黙の型変換が不可能な型の引数が指定された場合は，警告を出力し，強制的に型変換して組み込み関数のコードを生成します。

また，ldsr()/stsr() に対し，ハードウエアにないレジスタ番号を指定した場合は，エラーとします。
以下に，関数として記述できる命令を示します。

表 4.19 アセンブラ命令

アセンブラ命令	機能	組み込み関数
di	割り込み制御	void __DI(void);
ei		void __EI(void);
-	割り込みレベルの制御	void __set_il(割り込みの優先順位レベル , " 割り込み要求名 ");

-	割り込みレベルの制御	int NUM;
		void* ADDR 注 1;
		void __set_il_rh(NUM, ADDR);
		
		- NUM : 1 ～ 16
		movhi highw1(ADDR), r0, rX
		ld.b loww(ADDR)[rX], rY
		andi 0x00F0, rY, rY
		ori (優先順位 - 1), rY, rY
		st.b loww(ADDR)[rX]
		- NUM : 0
		movhi highw1(ADDR), r0, rX
		clr1 7, loww(ADDR)[rX]
		- NUM : -1
		movhi highw1(ADDR), r0, rX
		set1 7, loww(ADDR)[rX]
		- NUM : -2
		movhi highw1(ADDR), r0, rX
		clr1 6, loww(ADDR)[rX]
		- NUM : -3
		movhi highw1(ADDR), r0, rX
		set1 6, loww(ADDR)[rX]
		- NUM : -4 以下，および 17 以上
		範囲外エラー

nop	ノー・オペレーション	void __nop(void);
halt	プロセッサの停止	void __halt(void);
satadd	飽和加算	long a, b; long __satadd(a, b);
satsub	飽和減算	long a, b; long __satsub(a, b);
bsh	ハーフワード・データのバイト・スワップ	long a; long __bsh(a);
bsw	ワード・データのバイト・スワップ	long a; long __bsw(a);
hsw	ワード・データのハーフワード・スワップ	long a; long __hsw(a);
mul	符号つき乗算結果の上位 32 ビットを変数に代入する命令	long a, b; long __mul32(a, b);

アセンブラ命令	機能	組み込み関数
mulu	符号なし乗算結果の上位 32 ビットを変数に代入する命令	unsigned long a, b; unsigned long __mul32u(a, b);
sch0l	MSB 側からのビット（0）検索	long a; long __sch0l(a);
sch0r	LSB 側からのビット（0）検索	long a; long __sch0r(a);
sch1l	MSB 側からのビット（1）検索	long a; long __sch1l(a);
sch1r	LSB 側からのビット（1）検索	long a; long __sch1r(a);
ldsr	システム・レジスタへのロード	long regID; unsigned long a; void __ldsr(regID 注2, a);
ldsr	システム・レジスタへのロード	long regID; long selID; unsigned long a; void __ldsr_rh(regID 注2, selID 注2, a);
stsr	システム・レジスタの内容のストア	long regID; unsigned long __stsr(regID 注2);
stsr	システム・レジスタの内容のストア	long regID; long selID; unsigned long __stsr_rh(regID 注2, selID 注2);
caxi	比較と交換	long *a; long b, c; long __caxi(a, b, c);
clr1	ビット・クリア	unsigned char *a; int bit; void __clr1(a, bit);
set1	ビット・セット	unsigned char *a; int bit; void __set1(a, bit);
not1	ビット・ノット	unsigned char *a; int bit; void __not1(a, bit);
ldl.w 注3	アトミック・ロード	long *a; long __ldlw(a);
stc.w 注3	ストア	long *a; long b; void __stcw(a, b);
synce	例外同期化	void __synce(void);
synce	命令パイプライン同期化	void __synci(void);
syncm	メモリ同期化	void __syncm(void);
syncp	パイプライン同期化	void __syncp(void);
dbcp	PC トレース	void __dbcp(void);

アセンブラ命令	機能	組み込み関数
dbtag	タグ・トレース	const int a注4; void __dbtag(a);
dbpush	レジスタ値トレース (“dbpush rh-rt” 命令を利用)	const int regID_1注5; const int regID_2注5; void __dbpush(regID_1, regID_2);
	整数の値のトレース出力 (“dbpush rx” 命令を利用)	int a; void __traceivalue(a);
	浮動小数点の値のトレース出力 (“dbpush rx” 命令を利用)	float a; void __tracefvalue(a);
	文字列リテラルのトレース出力 (“dbpush rh-rt” 命令を利用)	const char *a注6; void __tracestring(a);

注 1. ADDR には割り込み制御レジスタのアドレスを指定してください。

注 2. regID にはシステム・レジスタ番号（0 ～ 31）を，selID には 0 ～ 31 を指定してください。

注 3. -Xcpu=g3k オプション使用時は警告を出力します。

注意 組み込み関数と同名の関数を定義して使用することはできません。
同名の関数を呼び出そうとしても，コンパイラが用意している組み込み関数処理を優先します。

4.2.4.8 構造体パッキング

CC-RH は，構造体メンバのアライメントを C 言語レベルで指定できます。この機能は，-Xpack オプションと同等ですが，構造体パッキング指令は C ソース内の任意の位置でアライメント値を指定できます。

注意 構造体をパッキングすると，データ領域を小さくできますが，プログラム・サイズは増え，実行速度も低下します。

(1) 構造体パッキングの形式
構造体パッキング機能は次の形式で指定します。

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

#pragma pack は，この指令が出現した時点で，構造体メンバのアライメント値に変更します。この数値をパッキング値と呼び，指定できる数値は，1，2，4 です。数値を指定しない場合，デフォルトのアライメントとなります。なお，この指令は出現した時点で有効となるので C ソース内に複数記述することができます。

例

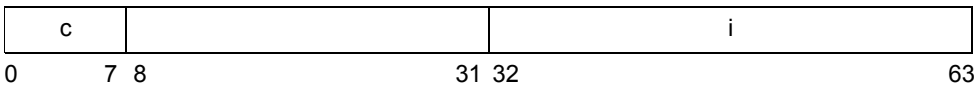
```
#pragma pack 1  /* 構造体メンバを 1 バイトのアライメントで整列 */
struct  TAG {
    char    c;
    int     i;
    short   s;
};
```

(2) 構造体パッキングのルール
構造体のメンバは，構造体のパッキング値とメンバの持つアライメント値の小さい方の値の整列条件を満たす形で並べられます。
たとえば，構造体のパッキング値が 2 のときメンバの形が int 型ならば 2 バイトの整列条件を満たす形で並べられます。

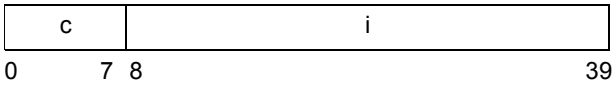
例

```
struct  S {
    char    c;      /*1 バイトの整列条件を満たす */
    int     i;      /*4 バイトの整列条件を満たす */
};

#pragma pack 1
struct  S1 {
    char    c;      /*1 バイトの整列条件を満たす */
    int     i;      /*1 バイトの整列条件を満たす */
};

#pragma pack 2
struct  S2 {
    char    c;      /*1 バイトの整列条件を満たす */
    int     i;      /*2 バイトの整列条件を満たす */
};

struct  S   sobj;   /* サイズ 8 バイト */
struct  S1  s1obj;  /* サイズ 5 バイト */
struct  S2  s2obj;  /* サイズ 6 バイト */
```

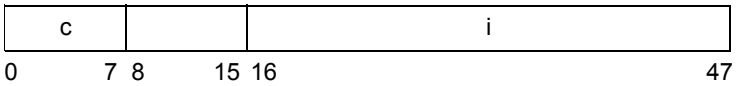
sobj



s1obj



s2obj



(3) 共用体

共用体をパッキングの対象として構造体パッキングと同様に扱います。

例 1.

```
union   U {
        char c;
        int i;
};

#pragma pack 1
union   U1 {
        char c;
        int i;
};

#pragma pack 2
union   U2 {
        char c;
        int i;
};

union   U   uobj;   /* サイズ 4 バイト */
union   U1  u1obj;  /* サイズ 4 バイト */
union   U2  u2obj;  /* サイズ 4 バイト */
```

例 2.

```
union   U {
    int i:7;
};

#pragma pack 1
union   U1 {
    int i:7;
};

#pragma pack 2
union   U2 {
    int i:7;
};

union   U   uobj;   /* サイズ 4 バイト */
union   U1  u1obj;  /* サイズ 1 バイト */
union   U2  u2obj;  /* サイズ 2 バイト */
```

(4) ビット・フィールド
ビット・フィールド要素の領域は次のように割り当てます

(a) 構造体のパッキング値がメンバの型の整列条件値と等しいあるいは大きい場合
構造体パッキング機能を利用しなかったときと同じように割り当てます。つまり，続けて割り当てるとその領域がメンバの型の整列条件を満たす境界を越えてしまう場合，その整列条件を満たしている領域から割り当てます。

(b) 構造体のパッキング値が要素の型の整列条件値より小さい場合

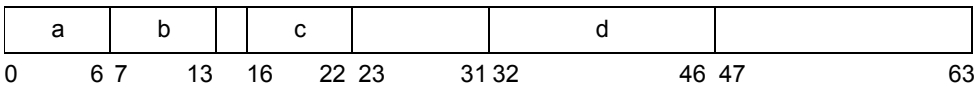
- 続けて割り当てるとその領域を含むバイト数が要素の型よりも大きくなる場合
  構造体のパッキング値の整列条件を満たす形で割り当てます。

- それ以外の場合
  続けて割り当てます。

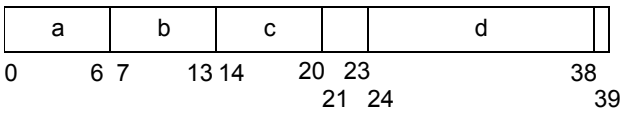
例

```
struct  S {
    short   a:7;    /*0 ～ 6 ビット目 */
    short   b:7;    /*7 ～ 13 ビット目 */
    short   c:7;    /*16 ～ 22 ビット目（2 バイト境界に整列）*/
    short   d:15;   /*32 ～ 46 ビット目（2 バイト境界に整列）*/
} sobj;

#pragma pack 1
struct  S1 {
    short   a:7;    /*0 ～ 6 ビット目 */
    short   b:7;    /*7 ～ 13 ビット目 */
    short   c:7;    /*14 ～ 20 ビット目 */
    short   d:15;   /*24 ～ 38 ビット目（バイト境界に整列）*/
} s1obj;
```

sobj



s1obj



(5) 構造体オブジェクトの先頭の整列条件
構造体オブジェクトの先頭の整列条件は，構造パッキング機能を利用しなかったときと同じです。

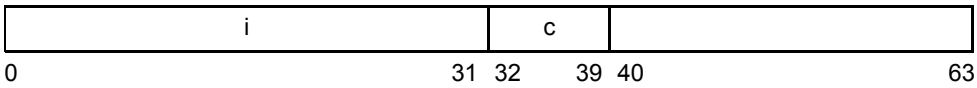
(6) 構造体オブジェクトのサイズ

構造体のサイズが構造体の整列条件の値と構造体のパッキング値の小さい方の値の倍数になるようにパッキングを行います。オブジェクトの先頭の整列条件は構造パッキング機能を利用しなかったときと同じです。

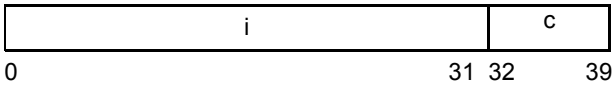
例 1.

```
struct  S {
    int     i;
    char    c;
};

#pragma pack 1
struct  S1 {
    int     i;
    char    c;
};

#pragma pack 2
struct  S2 {
    int     i;
    char    c;
};

struct  S   sobj;   /* サイズ 8 バイト */
struct  S1  s1obj;  /* サイズ 5 バイト */
struct  S2  s2obj;  /* サイズ 6 バイト */
```

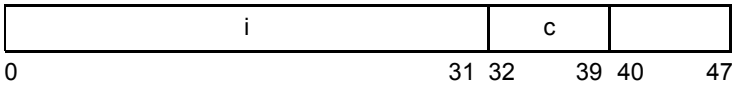
sobj



s1obj



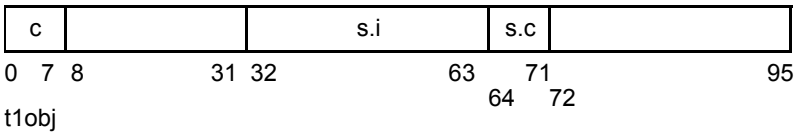
s2obj



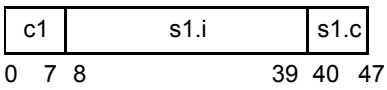
例 2.

```
struct  S {
    int     i;
    char    c;
};
struct  T {
    char    c;
    struct  S   s;
};

#pragma pack 1
struct  S1 {
    int     i;
    char    c;
};
struct  T1 {
    char    c;
    struct  S1  s1;
};

#pragma pack 2
struct  S2 {
    int     i;
    char    c;
};
struct  T2 {
    char    c;
    struct  S2  s2;
};

struct  T   tobj;   /* サイズ 12 バイト */
struct  T1  t1obj;  /* サイズ  6 バイト */
struct  T2  t2obj;  /* サイズ  8 バイト */
```

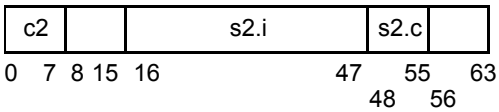
tobj



t1obj



t2obj



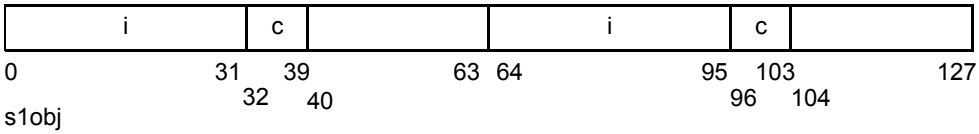
(7) 構造体配列のサイズ

構造体オブジェクトの配列のサイズは要素である構造体オブジェクトのサイズに要素数を乗算した値です。

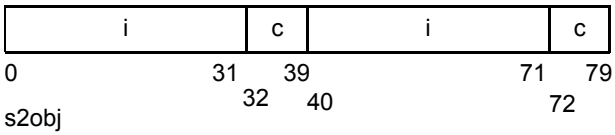
例

```
struct  S {
    int     i;
    char    c;
};

#pragma pack 1
struct  S1 {
    int     i;
    char    c;
};

#pragma pack 2
struct  S2 {
    int     i;
    char    c;
};

struct  S   sobj[2];    /* サイズ 16 バイト */
struct  S1  s1obj[2];   /* サイズ 10 バイト */
struct  S2  s2obj[2];   /* サイズ 12 バイト */
```

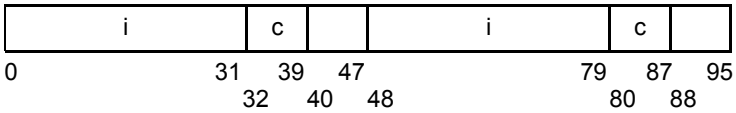
sobj



s1obj



s2obj



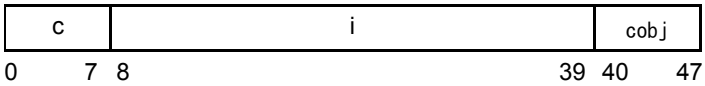
(8) オブジェクト間の領域

たとえば，次のソース・プログラムでは，sobj.c，sobj.i，cobj が隙間なく続いて配置される可能性があります（sobj，cobj の配置順は保証されません）。

例

```
#pragma pack 1
struct  S {
    char    c;
    int     i;
} sobj;
char    cobj;
```

sobj，cobj



(9) 構造体パッキング機能の注意点

(a) -Xpack オプションと #pragma pack 指令の同時指定について

C ソース中に #pragma pack 指令で構造体パッキング指定がある時に -Xpack オプションを指定した場合，最初の #pragma pack 指令が出現するまではオプション指定値がすべての構造体に適用されます。それ以降は #pragma pack 指令の値が適用されます。

その後に #pragma pack（値なし）を書くと，#pragma pack（値なし）以降の行は，オプション指定値が適用されます。

例 （-Xpack=4 を指定した場合）

```
struct  S2 {...};        /* オプションでパッキング値 4 を指定している
                            -Xpack=4 オプションが有効：パッキング値 4*/
#pragma pack 2           /*#pragma 指令でパッキング 2 を指定している
struct  S1 {...};           pragma pack(2) が有効：パッキング値 2*/
#pragma pack             /*#pragma 指令でパッキング値に指定なし
struct  S2_2 {...};         -Xpack=4 オプションが有効：パッキング値 4*/
```

(b) 構造体のパッキング値とメンバの持つアライメント値
構造体のメンバは，構造体のパッキング値とメンバの持つアライメント値の小さい方の値の整列条件を満たす形で並べられます。たとえば，構造体のパッキング値が 2 のときメンバの形が long 型ならば 2 バイトの整列条件を満たす形で並べられます。

例

```
struct S {
        char    c;      /*1 バイトの整列条件を満たす */
        long    i;      /*4 バイトの整列条件を満たす */
};

#pragma pack(1)
struct S1 {
        char    c;      /*1 バイトの整列条件を満たす */
        long    i;      /*1 バイトの整列条件を満たす */
};

#pragma pack(2)
struct S2 {
        char    c;      /*1 バイトの整列条件を満たす */
        long    i;      /*2 バイトの整列条件を満たす */
};

struct S        sobj;   /* サイズ 8 バイト */
struct S1       s1obj;  /* サイズ 5 バイト */
struct S2       s2obj;  /* サイズ 6 バイト */
```

(c) #pragma pack の入れ子指定
異なる #pragma pack 指定された構造体オブジェクトの入れ子は，次のように定義します。
アライメントが異なるメンバを持つ構造体・共用体メンバに対しては，警告を出します。
警告対象のメンバは，ソース・コードに書かれたとおりの #pragma pack に従います。

例

```
#pragma pack 1
struct ST1
{
        char    c;
#pragma pack 4
        struct ST4              //size=8, align=4 ( 型としては 4)
        {
                char    c;      //offset=1
                short   s;      //offset=3
                int     i;      //offset=5
        } st4;                  //size=8, align=1 (ST1 のメンバなので 1)
                                // メンバ st4 の位置で警告
        int     i;
} st1;                          //size=13, align=1
```

［注意事項］

pack=1 もしくは #pragma pack 1 を指定した構造体，共用体，クラスのメンバはポインタを用いてアクセスすることはできません（ポインタを使用したメンバ関数内でのアクセスを含みます)。

```
例
#pragma   pack  1
struct st {
    char  x;
    int  y;
} ST;
int *p = &ST.y; /* ST.y のアドレスが奇数になる場合があります */
void func(void){
    ST.y =1;    /* 正しくアクセスできます */
    *p = 1;     /* 正しくアクセスできない場合があります */
}
```

4.2.4.9 ビット・フィールドの割り付け

CC-RH は，ビット・フィールドの並び順の切り替えを指定できます。

(1) ビット・フィールドの割り付け指定の形式
ビット・フィールドの割り付けは次の形式で指定します。

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

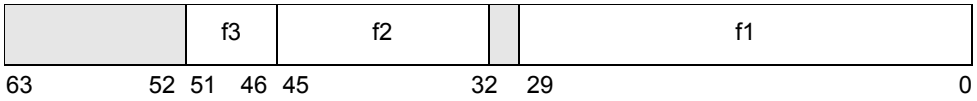
left を指定した場合は上位ビット側から，right を指定した場合は下位ビット側から，それぞれメンバが割り付けられます。
デフォルトは right となります。

例 1.

```
#pragma bit_order right
struct {
    unsigned long   f1:30;
    int             f2:14;
    unsigned int    f3:6;
} flag;
```

この例に示したビット・フィールドに対する内部表現は，次のようになります

図 4.11 ビット・フィールドの内部表現

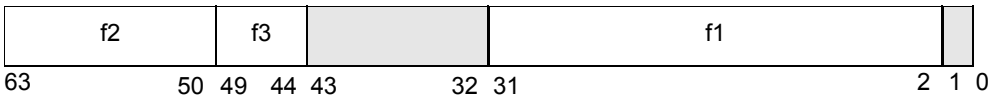


例 2.

```
#pragma bit_order left
struct {
    unsigned long   f1:30;
    int             f2:14;
    unsigned int    f3:6;
} flag;
```

この例に示したビット・フィールドに対する内部表現は，次のようになります。

図 4.12 ビット・フィールドの内部表現

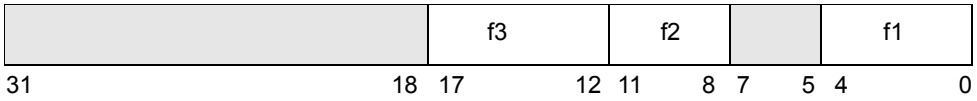


例 3.

```
#pragma bit_order right
struct {
    int             f1:5;
    char            f2:4;
    int             f3:6;
} flag;
```

この例に示したビット・フィールドに対する内部表現は，次のようになります

図 4.13 ビット・フィールドの内部表現

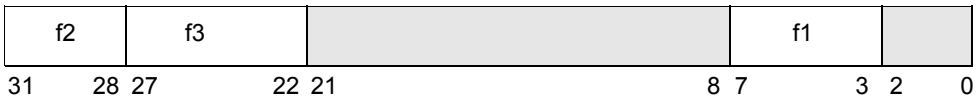


例 4.

```
#pragma bit_order left
struct {
    int             f1:5;
    char            f2:4;
    int             f3:6;
} flag;
```

この例に示したビット・フィールドに対する内部表現は，次のようになります

図 4.14 ビット・フィールドの内部表現



4.2.4.10 コア番号指定（マルチコア用）

コア番号指定機能とは，マルチコア利用時に，指定したコアのローカル・メモリや全コア共通のグローバル・メモリ上にデータを配置したり，指定したコアや任意のコアで関数を実行可能にする機能です。
ここで記述する #pragma とリンク・オプションを組み合わせて実現します。

例えば，ある変数 x（データ・セクションに配置されると仮定）をコア番号 1 のローカル・メモリに配置したい場合は次のようにします。

(a) 変数 x がファイル中，最も最初に定義・宣言されるより前に以下の pragma を置きます：

```
#pragma pmodule pm1
```

すると，コンパイラとアセンブラは変数 x をセクション .data.pm1 に配置します。

(b) ユーザは以下のリンク・オプションを指定します：

```
-start=.data.pm1/fe8f0000
```

すると，最適化リンカはセクション .data.pm1 をコア番号 1 のローカル・メモリ内に配置します（0xfe8f0000 はコア番号 1 のローカル・メモリ内のアドレスと仮定します）。

変数や関数にコア番号を指定することによって以下の利点が得られます。

- セクションにコア番号が付くことで，どの変数をどのコアのローカル・メモリに割り付けたか，どの関数をどのコアで実行するのか，といった対応付けを整理することができます。
  この情報は CS+ で表示することも可能です。
- デフォルト・セクションを含めたすべてのセクションにコア番号が付与されるので，コア毎にセクション名を変更する必要がなくなります。

- データや関数とコア番号との関係をコンパイラがチェックすることができます（コア 1 でのみ実行する関数からコア 2 でのみ実行する関数を呼んでいるとエラーを出力，など）。

(1) コア番号指定の形式
マルチコア利用時のコア指定は次の形式で指定します。

```
#pragma pmodule pm 指定
```

本 pragma は，-Xmulti_level オプションを指定した時のみ有効です。-Xmulti_level オプションを指定しない場合は，警告メッセージを出力して，コア番号指定を無視します。

pm 指定に書くことが可能な書式とその時の配置セクション名は次の表のとおりです。
pm 指定には pm1 ～ pm255，または cmn のみ書くことが可能です。pm 指定された変数および関数の配置セクション名の末尾には「.」と pm 指定した文字列が付加されます。

-Xmulti_level の値	pm 指定の値	意味	配置先セクション名
1	なし	デフォルト（cmn）を設定	***.cmn
	cmn	- データの場合 全コア共通のグローバル共有メモリに配置 - 関数の場合 どのコアで実行してもよい	***.cmn
	pm1	コア 1 用のデータ，または関数	***.pm1
	:	:	:
	pm255	コア 255 用のデータ，または関数	***.pm255
0	警告メッセージを出力して，コア番号指定を無視		*** （セクション末尾に文字列を付加しない。）

- cmn，pm1 ～ pm255 以外の pm 指定記述を行った場合は，コンパイル・エラーとなります。
- #pragma pmodule は，その宣言行以降に出現したすべての「静的変数の宣言」，「関数宣言」，「文字列リテラル[注]」に作用します。

  注 セクションに配置されるものすべてに作用するため，const に配置される文字列リテラルにも作用します。

- #pragma pmodule は，デフォルト・セクション名，およびユーザ指定セクション名の双方に，上に示した文字列を付加します。

  例 .data → .data.pm1
  mydata.data→ mydata.data.pm1

- 同一翻訳単位内で複数回出現する同一変数宣言，同一関数宣言に対して，異なる #pragma pmodule 指定がある場合，最初の指定を有効とします。

  例

```
#pragma pmodule pm1
extern  int     i;
#pragma pmodule pm2
int     i = 5;          //pm1 とする
```

次に，指定例を示します。
以下はすべて，-Xmulti_level=1 オプションを指定した時の #pragma 指定例です。

例 1.

```
#pragma section r0_disp16
int     i;                      //.zbss.pm1
--------------------------------------------------------------------------------
#pragma pmodule pm2
int     i;                      //.bss.pm2
int     j = 5;                  //.data.pm2
const int       k = 10;         //.const.pm2
void func(void)                 //.text.pm2
{
        func2("abcde");         //"abcde" は .const.pm2
}
--------------------------------------------------------------------------------
#pragma pmodule pm2
#pragma section r0_disp16
int     i;                      //.zbss.pm2
--------------------------------------------------------------------------------
#pragma section r0_disp16
#pragma pmodule pm2
int     i;                      //.zbss.pm2
--------------------------------------------------------------------------------
#pragma pmodule pm2
#pragma pmodule pm3
#pragma section r0_disp16
int     i;                      //.zbss.pm3
#pragma pmodule pm1
int     j;                      //.zbss.pm1
```

例 2.

```
extern  int     i;
#pragma pmodule pm2
int     i;                      //.bss.pm2（警告なし）
```

例 3.

```
#pragma pmodule pm2
extern  int     i;
int     i;                      //.bss.pm2（警告なし）
```

例 4.

```
#pragma pmodule pm2
extern  int     i;
#pragma pmodule pm3
int     i;                      //.bss.pm2（警告なし）
```

-

-

4.2.5 C ソースの修正

拡張機能を使用することにより，効率の良いオブジェクトを生成することができます。しかし，拡張機能は RH850 ファミリに則したもので，他に利用するためには修正が必要になる場合があります。

ここでは，他の C コンパイラから CC-RH への移植と，CC-RH から他の C コンパイラへの移植の 2 つの場合について，その方法を説明します。

＜他の C コンパイラから CC-RH ＞

- #pragma 注
  他の C コンパイラが #pragma をサポートしている場合は，C ソースを修正する必要があります。修正方法は，その C コンパイラの仕様によって検討します。

- 拡張仕様
  他の C コンパイラがキーワードを追加するなどの仕様の拡張を行っている場合は，修正する必要があります。修正方法はその C コンパイラの仕様によって検討します。

  注 ANSI でサポートされている前処理指令の 1 つで，#pragma に続く文字列をコンパイラへの指令として認識させるものです。その指令がコンパイラによってサポートされていなければ，#pragma 指令は無視され，コンパイルが続けられて正常に終了します。

＜ CC-RH から他の C コンパイラ＞

- CC-RH は，拡張機能としてキーワードの追加を行っているため，他の C コンパイラへ移植するためには，キーワードを削除するか，#ifdef で切り分けなければなりません。

例 1. キーワードを無効にする

```
#ifdef __CCRH__
#define interrupt        /*interrupt 関数を通常の関数にします */
#endif
```

例 2. 他の型に変更する

```
#ifdef __v850e3v5__
#define _Bool char       /*_Bool 型変数を char 型変数にします */
#endif
```

5. アセンブラ言語仕様

この章では，CC-RH がサポートするアセンブリ言語仕様について説明します。

5.1 ソースの記述方法

この節では，ソースの記述方法，式と演算子などについて説明します。

5.1.1 記述方法

アセンブリ言語文は，“ シンボル ”，“ ニモニック ”，“ オペランド ”，および “ コメント ” から構成されます。

```
[ シンボル ][: △ ]        [ ニモニック ]      [ オペランド ], [ オペランド ]      ;[ コメント ]
```

ラベルを記述する場合は，コロン，または 1 つ以上の空白で区切ります。ただし，コロンか空白かはニモニックで記述する命令によります。

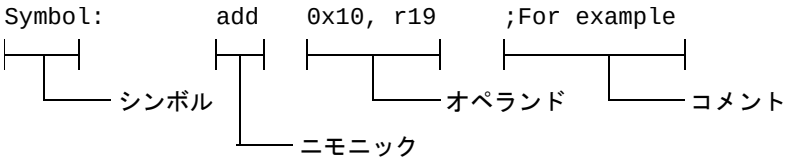
以下の箇所には，1 つ以上の空白があってもなくても問題ありません。

- シンボルとコロンの間
- コロンとニモニックの間
- 2 つ目以降のオペランドの前
- コメントの始まりのセミコロンの前

以下の箇所には，1 つ以上の空白が必要です。

- ニモニックとオペランドの間

図 5.1 アセンブリ言語文の構成



アセンブリ言語文は，1 行に 1 文を記述します。文の最後は改行（リターン）します。

(1) 文字セット
アセンブラがサポートするソース・プログラムで，使用できる文字は次の 3 つから構成されます。

- 言語文字
- 文字データ
- 注釈（コメント）用文字

(a) 言語文字
ソース上で命令を記述するために使用する文字です。

表 5.1 言語文字とその用途

文字	用途
数字	識別子，および定数の構成
英小文字（a ～ z）	ニモニック，識別子，および定数の構成
英大文字（A ～ Z）	識別子，および定数の構成
@	識別子，および定数の構成
_（アンダースコア）	識別子の構成

文字	用途
.（ピリオド）	識別子，および定数の構成
~	識別子，および定数の構成
,（カンマ）	オペランドの区切り
:（コロン）	ラベルの区切り
;（セミコロン）	コメントの開始
*	乗算演算子
/	除算演算子
+	正符号，および加算演算子
-（ハイフン）	負符号，および減算演算子
'（シングルクォーテーション）	文字定数，およびマクロのパラメータを 1 つにまとめる記号
<	比較演算子
>	比較演算子
( )	演算順序の指定
$	ソース・プログラム中の制御命令の前に付ける記号 相対アドレッシング開始記号 ラベルの gp オフセット参照
=	比較演算子
!	絶対アドレッシングの開始，および否定演算子
△（空白）	各欄の区切り記号
~	コンカティネート記号（マクロボディ内で使用）
&	論理積演算子
#	イミディエート値の開始，およびコメントの開始
[ ]	インダイレクト表示記号
"（ダブルクォーテーション）	文字列定数の開始と終了
%	ラベルの ep オフセット参照，および剰余演算子
<<	左論理シフト演算子
>>	右算術シフト演算子
|	論理和演算子
^	排他的論理和演算子

(b) 文字データ
文字データは，文字列定数，文字定数，および制御命令部を記述するために使用する文字です。

注意 0x00 を除くすべての文字（漢字かなを含みます。ただし，OS によってコードは異なります）が記述可能です。0x00 を記述するとエラーとなり，それ以降，引用符（ ’ ，または ”）で閉じるまで無視されます。

(c) 注釈（コメント）用文字
コメントを記述するために使用する文字です。

注意 文字データの文字セットと同一です。

(2) シンボル
シンボル欄には，シンボルを記述します。シンボルとは，数値データやアドレスなどに付けた名前のことです。

シンボルを使用することにより，ソースの内容がわかりやすくなります。

(a) シンボルの種類
シンボルは，その使用目的，定義方法によって，次に示す種類に分けられます。

シンボルの種類	使用目的	定義方法
ネーム	ソース中で，数値データやアドレスとして使用	シンボル定義疑似命令のシンボル欄に記述します。
ラベル	ソース中で，アドレス・データとして使用	シンボルのあとにコロン（:）を付けることにより定義します。
外部参照名	あるモジュールで定義されたシンボルをほかのモジュールで参照するときに使用	外部参照疑似命令のオペランド欄に記述します。
セクション名	リンク時に使用	セクション定義疑似命令のシンボル欄に定義します。
マクロ名	ソース中で，マクロ参照時に使用	マクロ疑似命令のシンボル欄に記述します。

(b) シンボル記述上の規則
シンボルは，次の規則に基づいて記述します。

- シンボルは，英数字，および英字相当文字（@，_，.）で構成します。
  ただし，先頭文字に数字（0 ～ 9）は使用できません。セクション名の先頭文字に数字を指定する場合は，セクション名をダブルクォーテーション（"）で囲んでください。
- シンボルの最大文字数は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。
- シンボルとして，予約語は使用できません。
  予約語については，「5.5 予 約 語」を参照してください。
- 同一シンボルを二度以上定義することはできません。
  ただし，.set 疑似命令で定義したシンボルは，.set 疑似命令で再定義することができます。
- アセンブラは，シンボルの大文字／小文字を区別します。
- シンボル欄にラベルを記述する場合は，ラベルの直後にコロン（:）を記述します。

正しいシンボルの例を以下に示します。

```
CODE01  .cseg                   ; “CODE01” はセクション名
VAR01   .set    0x10            ; “VAR01” はネーム
LAB01:  .dw     0               ; “LAB01” はラベル
```

誤ったシンボルの例を以下に示します。

```
1ABC    .set    0x3             ; 先頭文字に数字は使用できません。
LAB     mov     1, r10          ; “LAB” ラベルです。ニモニック欄とコロン（ : ）で区切ります。
FLAG:   .set    0x10            ; シンボルにはコロン（ : ）が必要ありません。
```

シンボルのみからなる文の例を以下に示します。

```
ABCD:                           ; ABCD がラベルとして定義されます。
```

(c) シンボルに関する注意事項
セクション定義疑似命令でセクション名が指定されなかったときは，アセンブラがセクション名を自動生成します。次に，そのセクションを示します。
同名で定義するとエラーとなります。

セクション名	疑似命令	再配置属性
.text	.cseg 疑似命令	TEXT
.const		CONST
.zconst		ZCONST
.zconst23		ZCONST23
.bss	.dseg 疑似命令	BSS
.data		DATA
.sbss		SBSS
.sdata		SDATA
.sbss23		SBSS23
.sdata23		SDATA23
.tdata		TDATA
.tbss4		TBSS4
.tdata4		TDATA4
.tbss5		TBSS5
.tdata5		TDATA5
.tbss7		TBSS7
.tdata7		TDATA7
.tbss8		TBSS8
.tdata8		TDATA8
.ebss		EBSS
.edata		EDATA
.ebss23		EBSS23
.edata23		EDATA23
.zbss		ZBSS
.zdata		ZDATA
.zbss23		ZBSS23
.zdata23		ZDATA23

(3) ニモニック

ニモニック欄には，インストラクションのニモニック，疑似命令，およびマクロ参照を記述します。
オペランドの必要なインストラクションや疑似命令，マクロ参照の場合，ニモニック欄とオペランド欄を 1 つ以上の空白，または TAB で区切ります。
ただし，インストラクションの第 1 オペランドの先頭が “#”，“$”，“！”，“[” の場合には，ニモニックと第 1 オペランドの間に何もなくても，正常にアセンブルが行われます。
正しい例を以下に示します。

```
mov     1, r10
```

誤った例を以下に示します。

```
mov1, r10        ; ニモニック欄とオペランド欄の間に，空白がありません。
mo v    1, r10   ; ニモニック中に空白があります。
MOVE             ; ニモニック欄に記述できない命令です。
```

(4) オペランド

オペランド欄には，インストラクションや疑似命令，およびマクロ参照の実行に必要なデータ（オペランド）を記述します。
各インストラクションや疑似命令により，オペランドを必要としないものや，複数のオペランドを必要とするものがあります。
2 個以上のオペランドを記述する場合には，各オペランドをコンマ（ , ）で区切ります。
オペランド欄に記述できるものは，次のものです。

- 定数（数値定数，文字定数，文字列定数）
- レジスタ名
- シンボル
- 式

なお，インストラクション・セットにおけるオペランドの表現形式と記述方法については，開発対象となる各デバイスのユーザーズ・マニュアルを参照してください。
以降に，オペランド欄に記述可能な各項目について説明します。

(a) 定数

定数は，それ自身で定まる値を持つもので，イミーディエト・データとも呼びます。
定数には，数値定数と文字定数，文字列定数があります。

<1> 数値定数

整定数として，2 進数，8 進数，10 進数，16 進数が記述可能です。
整定数，つまり，整数の定数は，32 ビット幅をもつ定数です。負の数は 2 の補数で表現されます。32 ビットで表現することのできる値を越える整数値が指定された場合，アセンブラはその整数値の下位 32 ビットの値を用いて処理を続行します（メッセージ等は出力しません）。

整定数の種類	表記方法	表記例
2 進数	数値の前に “0b” を記述	0b1101
8 進数	数値の前に “0” を記述	074
10 進数	数値をそのまま記述	128
16 進数	数値の前に “0x” を記述	0xA6

浮動小数点定数は次に示す要素で構成されます。指数値，および仮数値は 10 進定数で指定します。ただし，指数表現を用いない場合（3），（4），（5）は使用しません。

（1）仮数部の符号（+ は省略可）
（2）仮数部
（3）指数部を示す ’e’，または ’E’
（4）指数部の符号（+ は省略可）
（5）指数部

例

```
123.4
-100.
10e-2
-100.2E+5
```

仮数値の頭に ”0f”，または ”0F” を置くことにより浮動小数点定数であることを示すこともできます。

例

```
0f10
```

<2> 文字定数

文字定数は，1 つの文字をシングル・クォート “’” で囲むことにより構成され，囲まれた文字の値を示します。注

以下に示したエスケープ・シーケンスを “” と “” の間に指定した場合，1 つの文字を表すものとして扱われます。

例

```
'A'             ; 0x00000041
' '             ; 0x00000020（空白 1 個）
```

注　文字定数を指定した場合，その文字定数の値を持つ整定数を指定したものとみなして扱われます。

表 5.2　エスケープ・シーケンスの値と意味

エスケープ・シーケンス	値	意味
¥0	0x00	null 文字
¥a	0x07	アラート
¥b	0x08	バックスペース
\f	0x0C	フォーム・フィード
¥n	0x0A	改行
¥r	0x0D	キャリッジ・リターン
¥t	0x09	水平タブ
\v	0x0B	垂直タブ
\\	0x5C	バックスラッシュ
¥'	0x27	シングル・クォート
¥"	0x22	ダブル・クォート
¥?	0x3F	疑問符
¥ddd	0 ～ 0377	3 桁までの 8 進数（0 < d < 7）注
¥xhh	0 ～ 0xFF	2 桁までの 16 進数 （0 < h < 9，a < h < f，または A < h < F）

注　“¥377” を越える指定の場合，そのエスケープ・シーケンスの値は，下位 1 バイト分のみの値となります。0377 より大きい値にはなりません。たとえば，“¥777” の値は 0377 です。

<3> 文字列定数

文字列定数は，「(1) 文字セット」で示した文字を引用符（ " ）で囲んだものです。
引用符自体を文字列定数とする場合には，引用符を 2 個続けて記述します。
例を以下に示します。

```
"ab"            ; 0x6162
"A"             ; 0x41
" "             ; 0x20（空白 1 個）
```

(b) レジスタ名

オペランド欄に記述可能なレジスタとして，次のものがあります。

- r0，zero，r1，r2，hp，r3，sp，r4，gp，r5，tp，r6，r7，r8，r9，r10，r11，r12，r13，r14，r15，r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r27，r28，r29，r30，ep，r31，lp

r0 と zero（ゼロ・レジスタ），r2 と hp（ハンドラ・スタック・ポインタ），r3 と sp（スタック・ポインタ），r4 と gp（グローバル・ポインタ），r5 と tp（テキスト・ポインタ），r30 と ep（エレメント・ポインタ），r31 と lp（リンク・ポインタ）は同じレジスタを示します。

備考　PSW，およびシステム・レジスタは，ldsr/stsr 命令において，番号で指定します。なお，本アセンブラでは，PC をオペランドに指定する方法はありません。

(c) シンボル
アセンブラでは，命令，および疑似命令のオペランド指定で使用可能な絶対値式，または相対値式の構成要素として，シンボルを用いることができます。

(d) 式
式は，定数，またはシンボルを演算子で結合したものです。
インストラクションのオペランドとして数値表現可能なところに記述することができます。
式と演算子については，「5.1.2 式と演算子」を参照してください。
例を以下に示します。

```
TEN     .set    0x10
        mov     TEN - 0x05, r10
```

この記述例では，“TEN - 0x05” が式です。
この式は，シンボルと数値定数が -（マイナス）演算子で結合されています。式の値は “0x0B” です。
したがって，この記述は “mov 0x0B, r10” と書き換えることが可能です。

(5) コメント
コメント欄には，セミコロン（ ; ）のあとにコメント（注釈）を記述します。
コメント欄は，セミコロンからその行の改行コード，または EOF までです。
コメントを記述することにより，理解しやすいソースを作成できます。
コメント欄の記述は，機械語変換というアセンブル処理の対象とはならず，そのままアセンブル・リストに出力されます。
記述可能な文字は，「(1) 文字セット」に示すものです。

5.1.2 式と演算子

式とは，シンボル，定数，ロケーション・カウンタを示す $，前述の 3 つに演算子を付加したもの，または演算子で結合したものです。
式を構成する演算子以外の要素を項といい，記述された左側から順に第 1 項，第 2 項，… と呼びます。
演算子には「表 5.3 演算子の種類」に示すものがあり，演算実行上の優先順位が「表 5.4 演算子の優先順位」のように決められています。
演算の順序を変更するには，かっこ “ ( ) ” を使用します。
例を示します。

```
mov32   5 * (SYM + 1), r12
```

上記の例では，“5 * (SYM+1)” が式です。“5” が第 1 項，“SYM” が第 2 項，“1” が第 3 項です。“*”，“+”，“( )” が演算子です。

表 5.3 演算子の種類

演算子の種類	演算子
算術演算子	+，-，*，/，%，+ 符号，- 符号
論理演算子	!，&，|，^
比較演算子	==，!=，>，>=，<，<=，&&，||
シフト演算子	>>，<<
バイト分離演算子	HIGH，LOW
2 バイト分離演算子	HIGHW，LOWW，HIGHW1
セクション集合演算子	STARTOF，SIZEOF
その他の演算子	( )

上記の演算子は，単項演算子，2 項演算子に分けられます。

単項演算子	+ 符号，- 符号，!，HIGH，LOW，HIGHW，LOWW，HIGHW1
特殊演算子	DATAPOS，BITPOS

2 項演算子	+, -, *, /, MOD（%）, &, |, ^, ==, !=, >, >=, <, <=, >>, <<, &&, ||

表 5.4 演算子の優先順位

優先度	優先順位	演算子
高い	1	+ 符号，- 符号，!
	2	*, /, %, >>, <<
	3	&, |, ^
	4	+, -
	5	==, !=, >, >=, <, <=
低い	6	&&, ||

式の演算は，次の規則に従います。

- 演算の順序は，演算子の優先順位に従います。
  同一順位の場合は，左から右に演算されます。単項演算子の場合は，右から左に演算されます。
- かっこ“( )”の中の演算は，かっこの外の演算に先立って行われます。
- 式の演算は，符号なし 32 ビットで行います。
  演算中に 32 ビットを越えてオーバーフローした場合，オーバーフローした値は無視されます。
- 定数が 32 ビットを越える場合には，エラーとなり，その値は 0 とみなされて計算されます。
- 除算では，小数部分を切り捨てます。
  除算がゼロの場合は，エラーとなり，結果は 0 となります。
- 負の値は，2 の補数形式となります。
- 外部参照記号のアセンブル時の評価値はゼロです（評価値はリンク時に決定されます）。

表 5.5 式評価の例

式	評価値
2 + 4 * 5	22
(2 + 3) * 4	20
10/4	2
0 - 1	0xFFFFFFFF
-1 > 1	0x0（偽）
EXT 注 + 1	1

注 EXT：外部参照記号

5.1.3 算術演算子

算術演算子には，次のものがあります。

演算子	概要
+	第 1 項と第 2 項の値の加算
-	第 1 項と第 2 項の値の減算
*	第 1 項と第 2 項の値の乗算
/	第 1 項と第 2 項の値で剰余算を行い，整数部を求める
%	第 1 項と第 2 項の値で剰余算を行い，余りを求める
+ 符号	項の値をそのまま返す
- 符号	項の値の 2 の補数を求める

```
+
```

第 1 項と第 2 項の値の加算を行います。

［機能］

第 1 項と第 2 項の値の和を返します。

［使用例］

```
        .org    0x100
START:  jr      START + 6       ;  (1)
```

(1) jr 命令により，「“START” に割り付けられたアドレス＋ 6 番地」へジャンプします。
つまり，START ラベルが 0x100 番地の場合，“0x100 + 0x6 = 0x106” へジャンプします。

```
-
```

第 1 項と第 2 項の値の減算を行います。

[機能]

第 1 項と第 2 項の値の差を返します。

[使用例]

```
        .org    0x100
BACK:   jr      BACK - 6        ; (1)
```

(1) jr 命令により，「“BACK” に割り付けられたアドレス－ 6 番地」へジャンプします。
つまり，BACK ラベルが 0x100 番地の場合，“0x100 - 0x6 = 0xFA” へジャンプします。

```
*
```

第 1 項と第 2 項の値の乗算を行います。

[機能]

第 1 項と第 2 項の値の積を返します。

[使用例]

```
TEN     .set    0x10
        mov     TEN * 3, r10    ; (1)
```

(1) .set 疑似命令により，シンボル “TEN” に 0x10 という値が定義されます。
“TEN * 3” という式は “0x10 * 3” のことで，0x30 を返します。
したがって，“mov 0x30, r10” と記述することもできます。

```
/
```

第 1 項と第 2 項の値で剰余算を行い，整数部を求めます。

［機能］

第 1 項の値を第 2 項の値で割り，その値の整数部を返します。
小数部は切り捨てられます。
除数（第 2 項）が 0 の場合は，エラーとなります。

［使用例］

```
mov     256 / 50, r10           ; （1）
```

(1) “256 / 50 = 5　余り 6” となります。
よって，整数部の 5 を返します。
したがって，“mov 5, r10” と記述することもできます。

```
%
```

第 1 項と第 2 項の値で剰余算を行い，余りを求めます。

［機能］

第 1 項の値を第 2 項の値で割り，その値の余りを返します。
除数が 0 の場合は，エラーとなります。

［使用例］

```
mov     256 % 50, r10           ; （1）
```

(1) “256 / 50 = 5　余り 6” となります。
よって，余りの 6 を返します。
したがって，“mov 6, r10” と記述することもできます。

+ 符号

項の値をそのまま返します。

[機能]

項の値をそのまま返します。

[使用例]

```
FIVE    .set    +5              ; (1)
```

(1) 項の値 “5” をそのまま返します。
.set 疑似命令により，シンボル “FIVE” に 5 という値が定義されます。

- 符号

項の値の 2 の補数を求めます。

[機能]

項の値の 2 の補数をとった値を返します。

[使用例]

```
NO      .set    -1              ; (1)
```

(1) “-1” は 1 の 2 の補数となります。
0000 0000 0000 0000 0000 0000 0000 0001 の 2 の補数は
1111 1111 1111 1111 1111 1111 1111 1111
となります。
よって，.set 疑似命令により，シンボル “NO” に 0xFFFFFFFF が定義されます。

5.1.4 論理演算子

論理演算子には，次のものがあります。

演算子	概要
!	項のビットごとの論理否定を求める
&	第 1 項の値と第 2 項の値のビットごとの論理積を求める
|	第 1 項の値と第 2 項の値のビットごとの論理和を求める
^	第 1 項の値と第 2 項の値のビットごとの排他的論理和を求める

!

項のビットごとの論理否定を求めます。

[機能]

項のビットごとの論理否定をとり，その値を返します。

[使用例]

```
mov     !0x3, r10               ; (1)
```

(1) “0x3” の論理否定をとります。
よって，0xFFFC を返します。
したがって，“mov 0xFFFC, r10” と記述することもできます。

!)	0000	0000	0000	0000	0000	0000	0000	0011
	1111	1111	1111	1111	1111	1111	1111	1100

&

第 1 項の値と第 2 項の値のビットごとの論理積を求めます。

[機能]

第 1 項の値と第 2 項の値のビットごとの論理積をとり，その値を返します。

[使用例]

```
mov     0x6FA & 0xF, r10        ;  (1)
```

(1) “0x6FA” と “0xF” の論理積をとります。
よって，“0xA” を返します。
したがって，（1）は “mov 0xA, r10” と記述することもできます。

	0000	0000	0000	0000	0000	0110	1111	1010
&)	0000	0000	0000	0000	0000	0000	0000	1111
	0000	0000	0000	0000	0000	0000	0000	1010

```
|
```

第 1 項の値と第 2 項の値のビットごとの論理和を求めます。

[機能]

第 1 項の値と第 2 項の値のビットごとの論理和をとり，その値を返します。

[使用例]

```
mov     0xA | 0b1101, r10       ;  (1)
```

(1) “0xA” と “0b1101” の論理和をとります。
よって，“0xF” を返します。
したがって，（1）は “mov 0xF, r10” と記述することもできます。

	0000	0000	0000	0000	0000	0000	0000	1010
|	0000	0000	0000	0000	0000	0000	0000	1101
	0000	0000	0000	0000	0000	0000	0000	1111

```
^
```

第 1 項の値と第 2 項の値のビットごとの排他的論理和を求めます。

[機能]

第 1 項の値と第 2 項の値のビットごとの排他的論理和をとり，その値を返します。

[使用例]

```
mov32   0x9A ^ 0x9D, r10        ;  (1)
```

(1) “0x9A” と “0x9D” の排他的論理和をとります。
よって，“0x7” を返します。
したがって，（1）は “mov 0x7, r10” と記述することもできます。

	0000	0000	0000	0000	0000	0000	1001	1010
^)	0000	0000	0000	0000	0000	0000	1001	1101
	0000	0000	0000	0000	0000	0000	0000	0111

5.1.5 比較演算子

比較演算子には，次のものがあります。

演算子	概要
==	第 1 項の値と第 2 項の値が等しいかどうか比較
!=	第 1 項の値と第 2 項の値が等しくないかどうか比較
>	第 1 項の値が第 2 項の値より大きいかどうか比較
>=	第 1 項の値が第 2 項の値より大きい，または等しいかどうか比較
<	第 1 項の値が第 2 項の値より小さいかどうか比較
<=	第 1 項の値が第 2 項の値より小さい，または等しいかどうか比較
&&	第 1 項の値と第 2 項の値の論理積を求める
||	第 1 項の値と第 2 項の値の論理和を求める

==

第 1 項の値と第 2 項の値が等しいかどうか比較します。

［機能］

第 1 項の値と第 2 項の値が等しいときに ~0（真），等しくないときに 0（偽）を返します。

!=

第 1 項の値と第 2 項の値が等しくないかどうか比較します。

［機能］

第 1 項の値と第 2 項の値が等しくないときに ~0（真），等しいときに 0（偽）を返します。

```
>
```

第 1 項の値が第 2 項の値より大きいかどうか比較します。

［機能］

第 1 項の値が第 2 項の値より大きいときに ~0（真），等しいか小さいときに 0（偽）を返します。

```
>=
```

第 1 項の値が第 2 項の値より大きい，または等しいかどうか比較します。

［機能］

第 1 項の値が第 2 項の値より大きいか，等しいときに ~0（真），小さいときに 0（偽）を返します。

```
<
```

第 1 項の値が第 2 項の値より小さいかどうか比較します。

［機能］

第 1 項の値が第 2 項の値より小さいときに ~0（真），等しいか大きいときに 0（偽）を返します。

<=

第 1 項の値が第 2 項の値より小さい，または等しいかどうか比較します。

［機能］

第 1 項の値が第 2 項の値より小さいか等しいときに ~0（真），大きいときに 0（偽）を返します。

&&

第 1 項の値と第 2 項の値の論理積を求めます。

［機能］

第 1 項の論理値と第 2 項の論理値の論理積を求めます。

```
||
```

第 1 項の値と第 2 項の値の論理和を求めます。

［機能］

第 1 項の論理値と第 2 項の論理値の論理和を求めます。

5.1.6 シフト演算子

シフト演算子には，次のものがあります。

演算子	概要
>>	第 1 項の値を第 2 項で示す値分だけ右シフトした値を求める
<<	第 1 項の値を第 2 項で示す値分だけ左シフトした値を求める

```
>>
```

第 1 項の値を第 2 項で示す値分だけ右シフトした値を求めます。

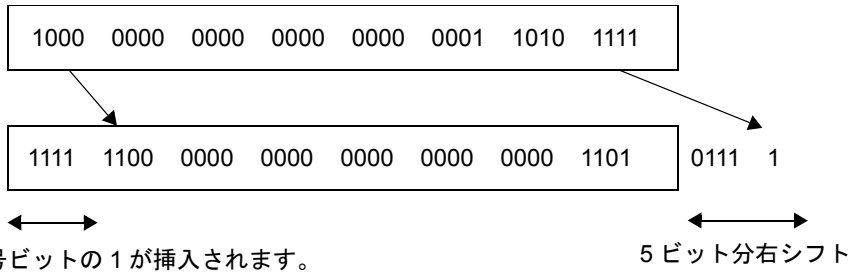
[機能]

第 1 項の値を第 2 項で示す値（ビット数）分だけ算術右シフトし，その値を返します。
符号ビットはシフトしません。
上位ビットには，シフトされたビット数だけ符号ビットの値が挿入されます。
シフト数が 0 の場合は，第 1 項の値がそのまま返されます。シフト数が 31 を越えた場合は，0 が返されます。

[使用例]

```
MOV32   0x800001AF >> 5, r20    ;（1）
```

(1) “0x800001AF” を符号ビットを残して 5 ビット分右シフトします。
よって，“0xFC00000D” を r20 に転送します。
したがって，“mov32 0xFC00000D, r20” と記述することもできます。



```
<<
```

第 1 項の値を第 2 項で示す値分だけ左シフトした値を求めます。

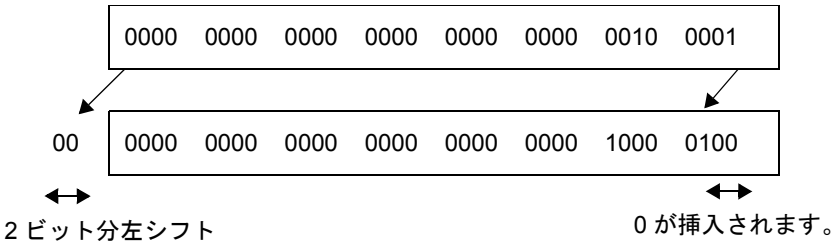
[機能]

第 1 項の値を第 2 項で示す値（ビット数）分だけ左シフトし，その値を返します。
下位ビットには，シフトされたビット数だけ 0 が挿入されます。
シフト数が 0 の場合は，第 1 項の値がそのまま返されます。シフト数が 31 を越えた場合は，0 が返されます。

[使用例]

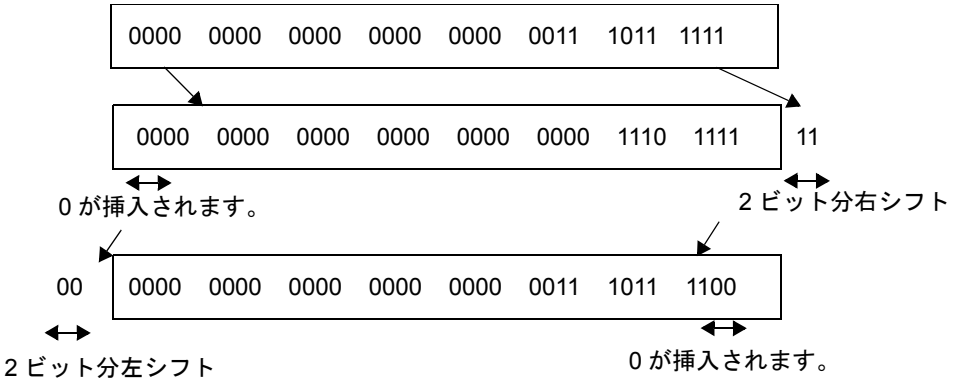
```
mov32   0x21 << 2,r20           ;  (1)
```

(1) “0x21” を 2 ビット分左シフトします。
よって，“0x84” を r20 に転送します。
したがって，“mov32 0x84,r20” と記述することもできます。




```
mov32   0x3BF >> 2 << 2, r20    ;  (2)
```

(2) “0x3BF” を 2 ビット分右シフトし，その後 2 ビット分左シフトします。
よって，“0x3BC” を r20 に転送します。
したがって，“mov32 0x3BC,r20” と記述することもできます。



5.1.7 バイト分離演算子

バイト分離演算子には，次のものがあります。

演算子	概要
HIGH	項の上位 8 ビットを求める
LOW	項の下位 8 ビットを求める

HIGH

項の上位 8 ビットを求めます。

[機能]

項の上位 8 ビットを返します。

[使用例]

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

(1) mov 命令実行により，0x1234 の上位 8 ビット値 0x12 を返します。
したがって，（1）は “mov 0x12, r10” と記述することもできます。

LOW

項の下位 8 ビットを求めます。

[機能]

項の下位 8 ビットを返します。

[使用例]

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

(1) mov 命令実行により，0x1234 の下位 8 ビット値 0x34 を返します。
したがって，（1）は “mov 0x34, r10” と記述することもできます。

5.1.8 2 バイト分離演算子

2 バイト分離演算子には，次のものがあります。

演算子	概要
HIGHW	項の上位 16 ビットを求める
LOWW	項の下位 16 ビットを求める
HIGHW1	項の上位 16 ビットに項の第 15 ビットの値を加算した値を求める

HIGHW

項の上位 16 ビットを求めます。

[機能]

項の上位 16 ビットを返します。

[使用例]

```
movea   HIGHW(0x12345678), R0, r10      ; (1)
```

(1) movea 命令実行により，0x12345678 の上位 16 ビット値 0x1234 を返します。
したがって，“movea 0x1234, R0, r10” と記述することもできます。

LOWW

項の下位 16 ビットを求めます。

[機能]

項の下位 16 ビットを返します。

[使用例]

```
movea   LOWW(0x12345678), R0, r10       ; (1)
```

(1) movea 命令実行により，0x12345678 の下位 16 ビット値 0x5678 を返します。
したがって，“movea 0x5678,R0, r10” と記述することもできます。

HIGHW1

項の上位 16 ビットに項の第 15 ビットの値を加算した値を求めます。

[機能]

項の上位 16 ビットに項の第 15 ビットの値を加算した値を返します。

[使用例]

```
movhi   HIGHW1(0x12348765), R0, r10     ; (1)
```

(1) movhi 命令実行により，0x12348765 の上位 16 ビット値 0x1234 に，0x12348765 の第 15 ビットの値 1 を加算した値 0x1235 を返します。
したがって，“movhi 0x1235, R0, r10” と記述することもできます。

5.1.9 セクション集合演算子

セクション集合演算子には，次のものがあります。

演算子	概要
STARTOF	項のセクションのリンク後の先頭アドレスを返す
SIZEOF	項のセクションのリンク後のサイズを返す

STARTOF

項のセクションのリンク後の先頭アドレスを返します。

[機能]

項のセクションのリンク後の先頭アドレスを返します。

[使用例]

```
.DW         STARTOF(.text)          ;  (1)
```

(1) 4 バイト領域を確保し，.text セクションの先頭アドレスで初期化します。

セクションの定義と整列条件の記述が必要です。

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

デフォルトの整列条件から変更していないセクションであれば整列条件の記述は省略できます。

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

デフォルトのセクション名，かつデフォルトの整列条件から変更していないセクションであれば，セクション定義，整列条件の記述は省略できます。

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

SIZEOF 演算子と組み合わせて指定する場合

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

数字で始まるセクション名を指定する場合

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

[注意事項]

- STARTOF は，データ定義疑似命令 .dw のオペランドにのみ記述できます。
- 同一モジュール内に定義していないセクション名を指定することはできません。
  そのため，必ずセクション定義疑似命令でセクションを定義し，.align 疑似命令で整列条件を合わせる必要があります。
  なお，セクション定義をしていない，および整列条件が異なる場合には，それぞれ次のメッセージが出力されます。
  - セクション定義をしていない場合：

    E0550249 : 構成に誤りがあります。

  - 整列条件が異なる場合：

    W0561322 : Section alignment mismatch : " *セクション名* "

- デフォルトのセクション名，および整列条件のデフォルト値を使用する場合は，セクション定義疑似命令，および .align 疑似命令の定義を省略することができます。

- 2 項演算子 + を使用して，SIZEOF 演算子と組み合わせて使用することができます。
  ただし，1 つのオペランドに STARTOF，および SIZEOF を複数記述すること，STARTOF，SIZEOF 以外の式を記述することはできません。

SIZEOF

項のセクションのリンク後のサイズを返します。

[機能]

項のセクションのリンク後のサイズを返します。

[使用例]

```
.DW     SIZEOF(.text)           ;  (1)
```

(1) 4 バイト領域を確保し，.text セクションのサイズで初期化します。

セクションの定義と整列条件の記述が必要です。

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

デフォルトの整列条件から変更していないセクションであれば整列条件の記述は省略できます。

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

デフォルトのセクション名，かつデフォルトの整列条件から変更していないセクションであれば，セクション定義，整列条件の記述は省略できます。

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

STARTOF 演算子と組み合わせて指定する場合

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

数字で始まるセクション名を指定する場合

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

[注意事項]

- SIZEOF は，データ定義疑似命令 .dw のオペランドにのみ記述できます。
- 同一モジュール内に定義していないセクション名を指定することはできません。
  そのため，必ずセクション定義疑似命令でセクションを定義し，.align 疑似命令で整列条件を合わせる必要があります。
  なお，セクション定義をしていない，および整列条件が異なる場合には，それぞれ次のメッセージが出力されます。
  - セクション定義をしていない場合：

    E0550249 : 構成に誤りがあります。

  - 整列条件が異なる場合：

    W0561322 : Section alignment mismatch : " *セクション名* "

- デフォルトのセクション名，および整列条件のデフォルト値を使用する場合は，セクション定義疑似命令，および .align 疑似命令の定義を省略することができます。

- 2 項演算子 + を使用して，STARTOF 演算子と組み合わせて使用することができます。
  ただし，1 つのオペランドに STARTOF，および SIZEOF を複数記述すること，STARTOF，SIZEOF 以外の式を記述することはできません。

5.1.10 その他の演算子

その他の演算子には，次のものがあります。

演算子	概要
( )	( ) 内の演算を優先して行う

( )

( ) 内の演算を優先して行います。

[機能]

( ) 内の演算を ( ) 外の演算に先立って行います。
演算の優先順位を変更したいときに使用します。
( ) が多重になっている場合は，一番内側の ( ) 内の式から演算します。

[使用例]

```
mov     (4 + 3) * 2, r10
```

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

(1)，(2) の順で演算を行い，14 という値を返します。
( ) がなければ

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

(1)，(2) の順で演算を行い，10 という値を返します。
演算子の優先順位については，「表 5.4 演算子の優先順位」を参照してください。

5.1.11 演算の制限

式は“定数”，“シンボル”，“ラベルの参照”，“演算子”，および“かっこ”を要素とし，これらの要素で構成される値を示します。式は，絶対値式と相対値式に分けて扱います。

(1) 絶対値式
定数値を示す式を“絶対値式”と呼びます。絶対値式は，命令においてオペランドを指定する場合，または疑似命令において値などを指定する場合に用いることができます。通常，絶対値式は，定数，またはシンボルによって構成されます。次に示した形式が絶対値式として扱われます。

(a) 定数式
定義済みのシンボル参照を指定した場合，そのシンボルに対して定義した値の定数が指定されたものとして扱われます。したがって，定数式は，定義済みのシンボル参照を，その構成要素として持つことができます。

例

```
SYM1    .set    0x100           ; シンボル SYM1 を定義
        mov     SYM1, r10       ; 定義済みの SYM1 は定数式として扱う
```

(b) シンボル
シンボルに関する式には，次のものがあります（“±” は “+” か “-” のどちらかになります）。

- シンボル
- シンボル ± 定数式
- シンボル - シンボル
- シンボル - シンボル ± 定数式

ここで言う“シンボル”とは，その時点において未定義なシンボル参照を指します。定義済みのシンボル参照を指定した場合は，そのシンボルに対して定義した値の“定数”が指定されたものとして扱います。

例

```
        add     SYM1 + 0x100, r11       ; この時点で SYM1 は未定義シンボル
SYM1    .set    0x10                    ; SYM1 を定義
```

(c) ラベル参照
ラベル参照に関する式には，次のものがあります（“±” は “+” か “-” のどちらかになります）。

- ラベル参照 - ラベル参照
- ラベル参照 - ラベル参照 ± 定数式

ラベル参照に関する式の例は，次のようになります。

例

```
mov     $label1 - $label2, r11
```

上記の例のような“2 つのラベル参照”は，次のように参照にする必要があります。

- 指定したファイル内で，同じセクションに定義をもつ
- 同じ参照方法（$label は $label 同士，#label は #label 同士など）

これらの条件を満たさない場合は，メッセージが出力され，アセンブルが中止されます。

ただし，現在のアセンブラの構成上，“ラベル参照に関する式”の中の “- ラベル参照” 側のラベル参照に，指定されたファイル内に定義を持たないラベルの絶対アドレス参照が指定された場合，他方のラベル参照の参照方法と同じ参照方法が用いられたものとみなして扱われます。なお，分岐命令に対して，この形式の絶対値式は指定できません。指定した場合は，メッセージが出力され，アセンブルが中止されます。
.DW 疑似命令においては，「2 つのラベル参照」が絶対アドレス参照であれば，別ファイルで異なるセクションに定義を持っていても，アセンブル可能とします。

(2) 相対値式
特定のアドレスからのオフセット値[注1]を示す式を“相対値式”と呼びます。相対値式は，命令においてオペランドを指定する場合，データ定義疑似命令において値を指定する場合に用いることができます。通常，相対値式は，ラベル参照によって構成されます。次に示した形式[注2]が相対値式として扱われます。

注 1. このアドレスはリンク時に定められます。このため，このオフセットの値もリンク時に定められます。

注 2. 絶対値式，相対値式ともに，“- シンボル + ラベルの参照 ” の形式の式を “ ラベルの参照 - シンボル ” の形式の式とみなすことはできますが，“ ラベルの参照 -（+ シンボル）” の形式の式を “ ラベルの参照 - シンボル ” の形式の式とみなすことはできません。このため，かっこ “( )” は定数式においてのみ用いるようにしてください。

(a) ラベル参照

ラベル参照に関する式には，次のものがあります（“±” は “+” か “-” のどちらかになります）。

- ラベル参照
- ラベル参照 ± 定数式
- ラベル参照 - シンボル
- ラベル参照 - シンボル ± 定数式

ラベル参照に関する式の例は次のようになります。

例

```
SIZE    .set    0x10
        add     #label1, r10
        add     #label1 + 0x10, r10
        add     #label2 - SIZE, r10
        add     #label2 - SIZE + 0x10, r10
```

5.1.12 識 別 子

識別子とは，シンボル，ラベル，マクロ名などに使用する名前です。
識別子は，次の規則に基づいて記述します。

- 識別子は，英数字，および英字相当文字（@，_，.）で構成します。
  ただし，先頭文字に数字（0 ～ 9）は使用できません。
- 識別子として，予約語は使用できません。
  予約語については，「5.5 予 約 語」を参照してください。
- アセンブラは，識別子の大文字／小文字を区別します

5.2 疑似命令

この節では，疑似命令について説明します。
疑似命令とは，アセンブラが一連の処理を行う際に必要な各種の指示を行うものです。

5.2.1 概　　要

インストラクションは，アセンブルの結果，機械語に変換されますが，疑似命令は，原則として機械語に変換されません。
疑似命令は，主に次の機能を持ちます。

- ソースの記述を容易にします。
- メモリの初期化や領域の確保を行います。
- アセンブラ，最適化リンカがその処理を行うために必要となる情報を与えます。

次に，疑似命令の種類を示します。

表 5.6　　疑似命令一覧

種類	疑似命令
セクション定義疑似命令	.cseg，.dseg，.section，.org
シンボル定義疑似命令	.set，.equ
コンパイラ出力疑似命令	.file，.line，.stack，._line_top，._line_end
データ定義，領域確保疑似命令	.db，.db2/.dhw，.dshw，.db4/.dw，.db8/.ddw，.float，.double，.ds，.align
外部定義，外部参照疑似命令	.public，.extern
マクロ疑似命令	.macro，.local，.rept，.irp，.exitm，.exitma，.endm

以降，各疑似命令について詳細な説明を行います。
説明の中で，[　] は大かっこの中が省略可能であることを，... は同一の形式を繰り返すことを示します。

5.2.2 セクション定義疑似命令

セクションとは，同種のルーチン，またはデータなどをブロック化したものです。セクション定義疑似命令は，セクションの開始，および終了を宣言するための疑似命令です。
セクションは，最適化リンカにおける配置単位です。

例

```
.cseg
  :
.dseg
  :
```

同じセクション名のセクション同士は，同一の再配置属性でなければなりません。したがって，再配置属性の異なる複数のセクションに，同一のセクション名を付けることはできません。同一のセクション名でありながら再配置属性が異なる場合はエラーとなります。
セクションは分割記述を行うことができます。つまり，1 つのソース・プログラム・ファイル内に記述した同一再配置属性，同一セクション名のセクションは，アセンブラ内部で連続したひとつのセクションとして処理を行います。
別々のソース・プログラム・ファイル内に分割記述した場合は，最適化リンカが処理を行います。
セクション名はシンボルとして参照できません。

セクション定義疑似命令には，次のものがあります。

表 5.7 セクション定義疑似命令

疑似命令	概要
.cseg	アセンブラにコード・セクションの開始を指示
.dseg	アセンブラにデータ・セクションの開始を指示
.section	アセンブラにセクションの開始を指示
.org	アセンブラに絶対アドレス形式セクションの開始を指示
.offset	セクション先頭からのオフセットを設定

.cseg

アセンブラにコード・セクションの開始を指示します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セクション名*]	.cseg	[*再配置属性*]	[; *コメント*]

[機能]

- .cseg 疑似命令は，アセンブラにコード・セクションの開始を指示します。
- .cseg 疑似命令以降に記述した命令は，再び 103.888 pt セクション定義疑似命令が現れるまでコード・セクションに属します。

[用途]

- .cseg 疑似命令で定義するコード・セクションには，インストラクションや .db，.dw 疑似命令等を記述します。
- サブルーチンなどの 1 つの機能を持つ単位の記述は，1 つのコード・セクションとして定義します。

[詳細説明]

- コード・セクションの開始アドレスは，.org 疑似命令により指定できます。
- 再配置属性とは，セクションの配置アドレスの範囲を限定するものです。
  次に，.cseg の再配置属性を示します。

表 5.8 .cseg の再配置属性

再配置属性	記述形式	説明	整列条件のデフォルト値
TEXT	TEXT	プログラムを配置します。	2
ZCONST	ZCONST	定数データ用（読み出し専用) のセクションで，r0 と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からプラス方向に最大 32K バイト）へ配置します。	4
ZCONST23	ZCONST23	定数データ用（読み出し専用）のセクションで，r0 と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からプラス方向に最大 4M バイト）に配置します。	4
CONST	CONST	定数データ用（読み出し専用）のセクションで，r0 と 32 ビットのディスプレースメントを用いて 2 命令で参照されるメモリ範囲（r0 からプラス方向に最大 4G バイト）に配置します。	4

- セクション定義疑似命令が出現する前に，ラベル，または機械語を出力する命令が記述された場合は，デフォルト・セクションとしてリロケータブルなコード・セクションを生成します。この時のセクション名は “.text”，再配置属性は “TEXT” となります。
- 「表 5.8 .cseg の再配置属性」以外の再配置属性を指定した場合は，アセンブラはエラーを出力します。
- .cseg 疑似命令のシンボル欄にセクション名を記述することにより，そのコード・セクションにシンボル（名前）を付けることができます。セクション名が省略されたコード・セクションには，アセンブラが自動的にデフォルトのセクション名を与えます。
  次に，.cseg のデフォルト・セクション名を示します。

再配置属性	デフォルト・セクション名
TEXT	.text
ZCONST	.zconst
ZCONST23	.zconst23
CONST	.const

デフォルトのセクション名は上記の再配置属性を持つものとし，別の再配置属性を指定することはできません。

- セクション名の先頭文字に数字を指定したい場合は，セクション名をダブルクォーテーション（"）で囲んでください。

例

```
test        .cseg   text
            nop
            nop
"test"      .cseg   text
            nop
            nop
"123test"   .cseg   text
            nop
            nop
```

- セクション名に指定可能な文字は下記です。
  - 英数字（0 ～ 9，a ～ z，A ～ Z）
  - 英字相当文字（@，_，.）

.dseg

アセンブラにデータ・セクションの開始を指示します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*セクション名*]	.dseg	[*再配置属性*]	[; *コメント*]

[機能]

- .dseg 疑似命令は，アセンブラにデータ・セクションの開始を指示します。
- .dseg 疑似命令以降，再びセクション定義疑似命令が現れるまで，データ・セクションに属します。

[用途]

- .dseg 疑似命令で定義するデータ・セクションには，主に .ds 疑似命令を記述します。

[詳細説明]

- データ・セクションの開始アドレスは，.org 疑似命令により指定できます。
- 再配置属性とは，データ・セクションの配置アドレスの範囲を限定するものです。
  次に，.dseg の再配置属性を示します。

表 5.9 .dseg の再配置属性

再配置属性	記述形式	説明	整列条件のデフォルト値
SDATA	SDATA	初期値を持ち，gp と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（SBSS セクションとあわせて最大 64K バイト）に配置します。	4
SBSS	SBSS	初期値を持たず，gp と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（SDATA セクションとあわせて最大 64K バイト）に配置します。	4
SDATA23	SDATA23	初期値を持ち，gp と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（SBSS23 セクションとあわせて最大 8M バイト）に配置します。	4
SBSS23	SBSS23	初期値を持たず，gp と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（SDATA23 セクションとあわせて最大 8MK バイト）に配置します。	4
TDATA	TDATA	初期値を持ち，ep を用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 256 バイト）に配置します。注	4
TDATA4	TDATA4	初期値を持ち，ep と 4 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 16 バイト）に配置します。	4
TBSS4	TBSS4	初期値を持たず，ep と 4 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 16 バイト）に配置します。	4

再配置属性	記述形式	説明	整列条件のデフォルト値
TDATA5	TDATA5	初期値を持ち，ep と 5 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 32 バイト）に配置します。	4
TBSS5	TBSS5	初期値を持たず，ep と 5 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 32 バイト）に配置します。	4
TDATA7	TDATA7	初期値を持ち，ep と 7 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 128 バイト）に配置します。	4
TBSS7	TBSS7	初期値を持たず，ep と 7 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 128 バイト）に配置します。	4
TDATA8	TDATA8	初期値を持ち，ep と 8 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 256 バイト）に配置します。	4
TBSS8	TBSS8	初期値を持たず，ep と 8 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（ep からプラス方向に最大 256 バイト）に配置します。	4
EDATA	EDATA	初期値を持ち，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（EBSS セクションとあわせて最大 64K バイト）に配置します。	4
EBSS	EBSS	初期値を持たず，ep と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（EDATA セクションとあわせて最大 64K バイト）に配置します。	4
EDATA23	EDATA23	初期値を持ち，ep と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（EBSS23 セクションとあわせて最大 8M バイト）に配置します。	4
EBSS23	EBSS23	初期値を持たず，ep と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（EDATA23 セクションとあわせて最大 8M バイト）に配置します。	4
ZDATA	ZDATA	初期値を持ち，r0 と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に ZBSS セクションとあわせて最大 32K バイト）に配置します。	4
ZBSS	ZBSS	初期値を持たず，r0 と 16 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に ZDATA セクションとあわせて最大 32K バイト）に配置します。	4
ZDATA23	ZDATA23	初期値を持ち，r0 と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に ZBSS23 セクションとあわせて最大 4M バイト）に配置します。	4
ZBSS23	ZBSS23	初期値を持たず，r0 と 23 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に ZDATA23 セクションとあわせて最大 4M バイト）に配置します。	4
DATA	DATA	初期値を持ち，r0 と 32 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に BSS セクションとあわせて最大 4G バイト）に配置します。	4

再配置属性	記述形式	説明	整列条件のデフォルト値
BSS	BSS	初期値を持たず，r0 と 32 ビットのディスプレースメントを用いて 1 命令で参照されるメモリ範囲（r0 からマイナス方向に DATA セクションとあわせて最大 4G バイト）に配置します。	4

注 TDATA 再配置属性のセクションを複数のソース・プログラムに定義した場合，リンク時にエラーになります。

- 「表 5.9 .dseg の再配置属性」以外の再配置属性を指定した場合は，アセンブラはエラーを出力します。
- 再配置属性に “BSS” の付くデータ・セクション中に機械語命令は記述できません。記述した場合はエラーとなります。
- .dseg 疑似命令のシンボル欄にセクション名を記述することにより，そのデータ・セクションにシンボル（名前）を付けることができます。セクション名が省略されたデータ・セクションには，アセンブラが自動的にデフォルトのセクション名を与えます。
  次に，.dseg のデフォルト・セクション名を示します。

再配置属性	デフォルト・セクション名
SDATA	.sdata
SBSS	.sbss
SDATA23	.sdata23
SBSS23	.sbss23
TDATA 注	.tdata
TDATA4	.tdata4
TBSS4	.tbss4
TDATA5	.tdata5
TBSS5	.tbss5
TDATA7	.tdata7
TBSS7	.tbss7
TDATA8	.tdata8
TBSS8	.tbss8
EDATA	.edata
EBSS	.ebss
EDATA23	.edata23
EBSS23	.ebss23
ZDATA	.zdata
ZBSS	.zbss
ZDATA23	.zdata23
ZBSS23	.zbss23
DATA	.data
BSS	.bss

注 TDATA 再配置属性に指定可能なセクション名は，デフォルト・セクション名のみです。ただし，省略は可能です。

デフォルトのセクション名は上記の再配置属性を持つものとし，別の再配置属性を指定することはできません。

- セクション名の先頭文字に数字を指定したい場合は，セクション名をダブルクォーテーション（"）で囲んでください。

例

```
test        .dseg   data
            .dw     0x1234
            .dw     0x5678
"test"      .dseg   data
            .dw     0x1234
            .dw     0x5678
"123test"   .dseg   data
            .dw     0x1234
            .dw     0x5678
```

- セクション名に指定可能な文字は下記です。
  - 英数字（0 ～ 9，a ～ z，A ～ Z）
  - 英字相当文字（@，_，.）

.section

アセンブラにセクションの開始を指示します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.section	*セクション名，再配置属性*	[; *コメント*]

[機能]

- .section 疑似命令は，アセンブラにセクション（コード，データの区別なし）の開始を指示します。

[用途]

- コード・セクション，データ・セクションにより，.cseg 疑似命令，.dseg 疑似命令を使い分けずに，.cseg 疑似命令，または .dseg 疑似命令で定義可能なすべてのセクションを .section 疑似命令をで定義することができます。

.org

絶対アドレス形式セクションの開始を指示します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.org	*絶対式*	[; *コメント*]

[機能]

- 絶対アドレス形式セクションの開始を指示します。
- .org 疑似命令以降，再びセクション定義疑似命令が現れるまで，有効です。
- .org 疑似命令を記述した行以降，再びセクション定義疑似命令が現れるまで，絶対アドレス形式セクションとします。
- 絶対アドレス形式セクションのセクション名は，“.org 疑似命令を記述したセクション ” ＋ “.AT” ＋ “ 指定アドレス ” となり，再配置属性は .org 疑似命令を記述したセクションの属性になります。
- ソース・プログラムの先頭でセクション定義疑似命令が出現する前に，.org 疑似命令が記述された場合は，セクション名は “.text.AT”+“ 指定アドレス ”，再配置属性は “TEXT” となります。

[使用例]

セクション定義疑似命令の直後に .org 疑似命令を記述した場合は，絶対アドレス形式セクションのみ生成します。

```
.section    "My_text", text
.org        0x12                    ;"My_text.AT12" を 0x12 に配置する
mov         r10, r11
.org        0x30                    ;"My_text.AT30" を 0x30 に配置する
mov         r11, r12
```

記述が直後でない場合は，.org 疑似命令の記述行以降を絶対アドレス形式セクションとして生成します。

```
.section    "My_text", text
nop                                 ;"My_text" に配置する
.org        0x50
mov         r10, r11                ;"My_text.AT50" に配置する
```

[注意事項]

- オペランドの値は「絶対値式」に従います。値として不正な記述をした場合はエラーとし，処理を終了します。
- 1 つのセクション定義中に，複数回記述できます。ただし，.org 疑似命令で指定したセクションのアドレスが，同一ファイル内で他の絶対アドレス形式セクションの配置範囲である場合エラーとなります。
- TDATA 再配置属性のセクションに .org 疑似命令を記述することはできません。記述した場合はエラーとし，処理を終了します。

.offset

セクション先頭からのオフセットを設定します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.offset	*絶対式*	[; *コメント*]

[機能]

- .offset 疑似命令を記述した行以降の命令コードが格納されるセクション先頭からのオフセットを設定します。
- .offset 疑似命令以降，再びセクション定義疑似命令が現れるまで，有効です。
- ソース・プログラムの先頭でセクション定義疑似命令が出現する前に，.offset 疑似命令が記述された場合は，セクション名は “.text”，再配置属性は “TEXT” となります。
- セクション名はダブルクォーテーション（"）で囲んで指定することもできます。
  セクション名の先頭文字に数字を指定したい場合は，セクション名をダブルクォーテーション（"）で囲んでください。

  例

```
.section    "My_data", data
.offset     0x12
mov         r10, r11            ; オフセットは 0x12 になる
```

[注意事項]

- オペランドの値は「絶対値式」に従います。値として不正な記述をした場合はエラーとし，処理を終了します。
- 1 つのセクション定義中に，複数回記述できます。ただし，.offset 疑似命令記述行のオフセットよりも小さい値を設定した場合はエラーとなります。
- .offset 疑似命令のオペランドに記述可能な値は，0x0 ～ 0x7fffffff までの範囲の絶対値式です。
  ただし，実際には動作するホスト・マシンのメモリ量に制限されます。

5.2.3 シンボル定義疑似命令

シンボル定義疑似命令は，ソース・モジュールを記述する際に使用するデータにシンボル（名前）を割り付けます。これにより，データ値の意味がはっきりし，ソース・モジュールの内容がわかりやすくなります。

シンボル定義疑似命令は，ソース・モジュール中で使用するシンボルの値をアセンブラに知らせるものです。

シンボル定義疑似命令には，次のものがあります。

表 5.10 シンボル定義疑似命令

疑似命令	概要
.set	ネームの定義
.equ	シンボルの定義

.set

ネームを定義します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
ネーム	.set	*絶対式*	[; *コメント*]

[機能]

シンボル欄に指定したネーム名と，オペランド欄に指定した絶対式の値を持つネームを定義します。

[用途]

- ソース・プログラム中で使用する数値データをネームとして定義し，機械語命令や疑似命令のオペランドに数値データの代わりに記述できます。
- ソース・プログラム中で頻繁に使用する数値データはネームとして定義しておくと，ソース・プログラム中のあるデータ値を変更しなければならない場合に，そのネームのオペランド値を変更するだけですみます。

[詳細説明]

- オペランドの記述形式にエラーがある場合，処理を終了します。
- .set 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- ネームは再定義可能なシンボルです。
- ネームを外部定義することはできません。

[使用例]

ネーム sym1 の値を 0x10 として定義する。

```
sym1    .set 0x10
```

[注意事項]

- 値の指定にラベル参照や，その時点において未定義なシンボル参照を用いることはできません。
  値の指定にラベル参照や，その時点において未定義なシンボル参照を用いた場合，次のメッセージが出力され，アセンブルが終了されます。

  E0550203 : 式の要素 *string* に誤りがあります。

- ラベル名，.macro 疑似命令で定義済みのマクロ名，およびマクロの仮パラメータと同名のシンボルを指定した場合，次のメッセージが出力され，アセンブルが終了されます。

  E0550212 : 指定されたシンボルはすでに *string* として定義されています。

.equ

シンボルを定義します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
シンボル	.equ	*絶対式*	[; *コメント*]

[機能]

シンボル欄に指定したシンボル名と，オペランド欄に指定した絶対式の値を持つシンボルを定義します。

[用途]

- ソース・プログラム中で使用する数値データをシンボルとして定義し，機械語命令や疑似命令のオペランドに数値データの代わりに記述できます。

[詳細説明]

- オペランドの記述形式にエラーがある場合，処理を終了します。
- .equ 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- すでに定義されたシンボルは再定義することはできません。
- .equ 疑似命令で生成したシンボルは .public 疑似命令で外部定義することができます。

5.2.4 コンパイラ出力疑似命令

コンパイラ出力疑似命令は，コンパイラのデバッグ情報などコンパイラが出力する情報をアセンブラに知らせるものです。
コンパイラ出力疑似命令には，次のものがあります。

表 5.11 コンパイラ出力疑似命令

疑似命令	概要
.file	シンボル・テーブル・エントリを生成
.line	C ソース・プログラムの行番号情報
.stack	シンボルに対するスタック使用量を定義
._line_top	#pragma inline_asm 指定情報
._line_end	#pragma inline_asm 指定情報

.file

シンボル・テーブル・エントリを生成（file タイプ）します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.file	" *ファイル名* "	[; *コメント*]

[機能]

- .file 疑似命令はコンパイラのデバッグ情報です。

[詳細説明]

- ファイル名は，指定イメージで記述します。
- コンパイラが出力する C ソース・プログラムのファイル名です。

.line

C ソース・プログラムの行番号情報です。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.line	[" *ファイル名* ",] *行番号*	[; *コメント*]

[機能]

- .line 疑似命令はコンパイラのデバッグ情報です。

[詳細説明]

- デバッグ時に参照する行番号とファイル名を変更します。
- ソース・プログラム内の最初の .line 疑似命令 以降，次の .line 疑似命令まで行番号とファイル名を更新しません。
- ファイル名を省略した場合は，行番号だけを変更します。
- .line 疑似命令は，コンパイラが出力する C ソース・プログラムの行番号情報です。

.stack

シンボルに対するスタック使用量を定義します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.stack	*シンボル名 = 値*	[; *コメント*]

[機能]

- .stack 疑似命令はコンパイラのデバッグ情報です。

[詳細説明]

- シンボルに対するスタック使用量を定義します。
- 1 つのシンボルに対して定義できるスタック使用量は 1 度のみとし，以降の定義は無視されます。
- 指定可能なスタック使用量は，0x0 ～ 0xFFFFFFFC の範囲の 4 の倍数のみとし，それ以外を指定した場合はその定義を無効となります。
- .stack 疑似命令は，コンパイラが出力する C ソース・プログラムの関数情報です。

._line_top

コンパイラの #pragma inline_asm 指定情報です。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	._line_top	inline_asm	[; *コメント*]

[機能]

- ._line_top 疑似命令はコンパイラの #pragma inline_asm 指定情報です。

[詳細説明]

- ._line_top 疑似命令は，コンパイラが出力する C ソース・プログラムの #pragma inline_asm 指定情報です。
- ._line_top 疑似命令は，inline_asm 指定関数における命令列の開始を意味します。

[注意事項]

- inline_asm 指定関数では，アセンブラ制御命令の記述を許可しません。また疑似命令については，以下に記載した命令以外の記述はできません。下記以外の命令を記述した場合はエラーとなります。
  - データ定義疑似命令（.db/.db2/.dhw/.db4/.dw/.db8/.ddw/.dshw/.ds/.float/.double）
  - マクロ疑似命令（.macro/.irp/.rept/.local）
- inline_asm 指定関数では，ローカル・ラベル以外のラベル定義は禁止とし，記述した場合はエラーとなります。

._line_end

コンパイラの #pragma inline_asm 指定情報です。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	._line_end	inline_asm	[; *コメント*]

[機能]

- ._line_end 疑似命令はコンパイラの #pragma inline_asm 指定情報です。

[詳細説明]

- ._line_end 疑似命令は，コンパイラが出力する C ソース・プログラムの #pragma inline_asm 指定情報です。
- ._line_end 疑似命令は，inline_asm 指定関数における命令列の終了を意味します。

[注意事項]

- inline_asm 指定関数では，アセンブラ制御命令の記述を許可しません。また疑似命令については，以下に記載した命令以外の記述はできません。下記以外の命令を記述した場合はエラーとなります。
  - データ定義疑似命令（.db/.db2/.dhw/.db4/.dw/.db8/.ddw/.dshw/.ds/.float/.double）
  - マクロ疑似命令（.macro/.irp/.rept/.local）
- inline_asm 指定関数では，ローカル・ラベル以外のラベル定義は禁止とし，記述した場合はエラーとなります。

5.2.5 データ定義，領域確保疑似命令

データ定義疑似命令は，プログラムで使用する定数データを定義します。
定義したデータの値は，機械語として生成されます。
領域確保疑似命令は，プログラムで使用するメモリの領域を確保します。
データ定義，領域確保疑似命令には，次のものがあります。

表 5.12 データ定義，領域確保疑似命令

疑似命令	概要
.db	1 バイト領域を初期化
.db2/.dhw	2 バイト領域を初期化
.dshw	2 バイトの領域を，指定した値を右に 1 ビット・シフトした値で初期化
.db4/.dw	4 バイト領域を初期化
.db8/.ddw	8 バイト領域を初期化
.float	4 バイト領域を初期化
.double	8 バイト領域を初期化
.ds	オペランドで指定したバイト数分のメモリ領域を確保
.align	ロケーション・カウンタの値を整列

.db

1 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.db	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.db	*式* [, ...]	[; *コメント*]
[*ラベル* :]	.db	" *文字列定数* "	[; *コメント*]

[機能]

- 1 バイト領域を初期化します。
  初期化するバイト数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリをバイト単位で初期化します。

[用途]

- プログラムで使用する式や文字列を定義するときに，.db 疑似命令を使用します。

[詳細説明]

- オペランドがカッコ “ (”，“) ” で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは，指定されたバイト数分の領域を “0” で初期化します。

(b) サイズには，絶対式を記述できます。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 式
式の値は 1 バイトのデータとして確保されます。したがって，式の値は 0x0 ～ 0xFF の間でなければなりません。1 バイトを越えた場合，下位 1 バイトがデータとして確保されます。

(b) 文字列定数
第 1 オペランドが対応する ’“’ で囲まれているときは文字列定数を記述したとみなします。
文字列定数が記述された場合，1 文字に対して，それぞれ 8 ビット ASCII コードが確保されます。

- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式が記述することができます。
- .db 疑似命令を記述するセクションの再配置属性に “BSS” が付く場合には初期値指定はできないものとして，エラーを出力します。

[使用例]

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

(1) サイズを指定しているので，それぞれのバイト領域を値 “0” で初期化します。

(2) 6 バイトの領域を文字列 “ABCDEF” で初期化します。

(3) 3 バイトの領域を 0xA，0xB，0xC で初期化します。

(4) 4 バイトの領域を 0x0 で初期化します。

(5) この記述はエラーとなります。

.db2/.dhw

2 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.db2	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.db2	*式* [, ...]	[; *コメント*]
[*ラベル* :]	.dhw	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.dhw	*式* [, ...]	[; *コメント*]

[機能]

- 2 バイト領域を初期化します。
  初期化する 2 バイト・データ数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリを 2 バイト単位に初期化します。

[用途]

- プログラムで使用するアドレスやデータなどの 2 バイトの定数を定義するときに，.db2，.dhw 疑似命令を使用します。

[詳細説明]

- オペランドがカッコ “ (”，“) ” で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは指定された 2 バイト・データ数分の領域を “0” で初期化します。

(b) サイズには，絶対式を記述できます。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 式
式の値は，2 バイト・データとして確保されます。したがって，式の値は 0x0 ～ 0xFFFF の間でなければなりません。2 バイトを越えた場合，下位 2 バイトがデータとして確保されます。
文字列定数は，初期値として記述できません。

- .db2，.dhw 疑似命令を記述するセクションの再配置属性に “BSS” が付く場合には初期値指定はできないものとして，エラーを出力します。
- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式が記述することができます。

.dshw

2 バイトの領域を，指定した値を右に 1 ビット・シフトした値で初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.dshw	*式* [, ...]	[; *コメント*]

[機能]

- 2 バイトの領域を，指定した値を右に 1 ビット・シフトした値で初期化します。

[詳細説明]

- 式の値を右に 1 ビット・シフトした値を 2 バイトデータとして確保します。
- .dshw 疑似命令はセクションの再配置属性に “BSS” が付く場合には記述できないものとし，エラーを出力します。
- オペランドの式には絶対式が記述可能です。
- 式の値は，右に 1 ビット・シフトした値が 0x0 ～ 0xFFFF の範囲でなければなりません。それ以外の場合は，下位 2 バイト分のデータを確保します。
- 式は，カンマで区切ることにより，1 行の範囲ならばいくつでも指定できます。
- オペランドに文字列定数を記述することはできません。

.db4/.dw

4 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.db4	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.db4	*式* [, ...]	[; *コメント*]
[*ラベル* :]	.dw	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.dw	*式* [, ...]	[; *コメント*]

[機能]

- 4 バイト領域を初期化します。
  初期化する 4 バイト・データ数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリを 4 バイト単位に初期化します。

[用途]

- プログラムで使用するアドレスやデータなどの 4 バイトの定数を定義するときに，.db4，.dw 疑似命令を使用します。

[詳細説明]

- オペランドがカッコ “ (”，“) ” で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは指定された 4 バイト・データ数分の領域を “0” で初期化します。

(b) サイズには，絶対式を記述できます。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 式
式の値は，4 バイト・データとして確保されます。したがって，式の値は 0x0 ～ 0xFFFFFFFF の間でなければなりません。4 バイトを越えた場合，下位 4 バイトがデータとして確保されます。
文字列定数は，初期値として記述できません。

- 初期値は，1 行の範囲であれば複数指定することができます。
- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式が記述することができます。
- .db4，.dw 疑似命令を記述するセクションの再配置属性に “BSS” が付く場合には初期値指定はできないものとして，エラーを出力します。

.db8/.ddw

8 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.db8	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.db8	*絶対式* [, ...]	[; *コメント*]
[*ラベル* :]	.ddw	(*絶対式*)	[; *コメント*]
[*ラベル* :]	.ddw	*絶対式* [, ...]	[; *コメント*]

[機能]

- 8 バイト領域を初期化します。
  初期化する 8 バイト・データ数は，サイズとして指定することができます。
- オペランドで指定された初期値で，メモリを 8 バイト単位に初期化します。

[用途]

- プログラムで使用するアドレスやデータなどの 8 バイトの定数を定義するときに，.db8，.ddw 疑似命令を使用します。

[詳細説明]

- オペランドがカッコ “ (”，“) ” で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは指定された 8 バイト・データ数分の領域を “0” で初期化します。

(b) サイズには，絶対式を記述できます。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 式
式の値は，8 バイト・データとして確保されます。したがって，式の値は 0x0 ～ 0xFFFFFFFFFFFFFFFF の間でなければなりません。8 バイトを越えた場合，下位 8 バイトがデータとして確保されます。
文字列定数は，初期値として記述できません。

- .db8，.ddw 疑似命令はセクションの再配置属性に “BSS” が付く場合には記述できないものとし，エラーを出力します。
- 初期値は，1 行の範囲であれば複数指定することができます。

.float

4 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.float	*絶対式* [, ...]	[; *コメント*]

[機能]

- 4 バイト領域を初期化します。
- オペランドで指定された絶対式で，メモリを 4 バイト単位に初期化します。

[詳細説明]

- 絶対式の値は，単精度浮動小数点数として確保されます。したがって，式の値は 1.40129846e-45 ～ 3.40282347e+38 の間でなければなりません。それ以外の場合は，下位 4 バイト分のデータが単精度浮動小数点数として確保されます。
- .float 疑似命令はセクションの再配置属性に “BSS” が付く場合には記述できないものとし，エラーを出力します。
- 絶対式は，1 行の範囲であれば複数指定することができます。

.double

8 バイト領域を初期化します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.double	*絶対式* [, ...]	[; *コメント*]

[機能]

- 8 バイト領域を初期化します。
- オペランドで指定された初期値で，メモリを 8 バイト単位に初期化します。

[詳細説明]

- 絶対式の値は，倍精度浮動小数点数として確保されます。したがって，式の値は 4.9406564584124654e-324 ～ 1.7976931348623157e+308 の間でなければなりません。それ以外の場合は，下位 8 バイト分のデータが倍精度浮動小数点数として確保されます。
- .double 疑似命令はセクションの再配置属性に “BSS” が付く場合には記述できないものとし，エラーを出力します。
- 絶対式は，1 行の範囲であれば複数指定することができます。

.ds

オペランドで指定したバイト数分のメモリ領域を確保します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.ds	(*絶対式*)[, ...]	[; *コメント*]
[*ラベル* :]	.ds	*絶対式*	[; *コメント*]

[機能]

- オペランドで指定したバイト数分のメモリ領域を確保します。

[用途]

- .ds 疑似命令は，主にプログラムで使用するメモリ（RAM）の領域を確保するときに使用します。
  ラベルがある場合は，確保したメモリ領域の先頭アドレスの値をそのラベルに割り付けます。ソース・モジュールでは，このラベルを使用してメモリを操作する記述をします。

[詳細説明]

- 第 1 オペランドがカッコ “ (”，“) ” で囲まれている場合はサイズ指定とみなされ，そうでない場合は初期値とみなされます。
- 第 1 オペランドがサイズ指定であり，さらに第 2 オペランドが指定されているときはその値を初期値指定とみなされます。

(1) サイズ指定の場合

(a) オペランドにサイズを記述した場合，アセンブラは指定されたバイト数分の領域を，初期値指定があればその値で，初期値指定がなければ “0” で初期化します。ただし，サイズ指定のバイト数が 0 の場合は，領域の確保は行われません。

(b) サイズには，絶対式を記述します。サイズの記述が不正な場合，エラーが出力され，初期化は行われません。

(2) 初期値指定の場合

(a) 式
式の値は，バイト・データとして確保されます。式の値は 0x0 ～ 0xFF の間でなければなりません。それ以外の場合，下位 1 バイトがデータとして確保され，エラーが出力されます。

- 初期値として，リロケータブルなシンボルや外部参照シンボルを含んだ式が記述することができます。
- 本疑似命令を記述するセクションの再配置属性に “BSS” が付く場合には初期値指定はできないものとして，エラーを出力し，本疑似命令を無視します。

.align

ロケーション・カウンタの値を整列します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.align	*整列条件* [, *絶対式*]	[; *コメント*]

[機能]

- ロケーション・カウンタの値を整列します。

[詳細説明]

- 前に指定されたセクション定義疑似命令によって指定される現在のセクションに対するロケーション・カウンタ値を，第 1 オペランドで指定した整列条件で整列します。なお，ロケーション・カウンタ値を整列したことによりホールが生じた場合，生じたホールを第 2 オペランドで指定した絶対式の値，またはデフォルト値の 0 で埋めます。
- 整列条件は 2 以上 2^{31} 未満の偶数にしてください。それ以外のものを指定した場合，エラーが出力され，アセンブルが中止されます。
- 第 2 オペランドの絶対式の値は 0x0 ～ 0xFF の間でなければなりません。それ以上の値を指定した場合，下位 1 バイト分の値が用いられます。
- 本疑似命令は，そのセクションに対する指定したファイル内でのロケーション・カウンタ値を整列するだけであり，配置後のアドレスを整列するものでもありません。

5.2.6 外部定義，外部参照疑似命令

外部定義，外部参照疑似命令は，ほかのモジュールで定義されているシンボルを参照する場合に，その関連性を明白にさせるためのものです。

1 つのプログラムがモジュール 1 とモジュール 2 に分けて作成されている場合を考えます。モジュール 1 中において，モジュール 2 中で定義されているシンボルを参照したい場合，お互いのモジュールで何の宣言もなくそのシンボルを使うわけにはいきません。このため，「使いたい」，「使ってもよい」の表示をそれぞれのモジュールで行う必要があります。

モジュール 1 では，「ほかのモジュール中で定義されているシンボルを参照したい」というシンボルの外部参照宣言をします。一方，モジュール 2 では，「そのシンボルは，ほかのシンボルで参照してもよい」というシンボルの外部定義宣言をします。

外部参照と外部定義という 2 つの宣言が有効に行われて，はじめてそのシンボルを参照することができます。

この相互関係を成立させるのが，外部定義，外部参照疑似命令であり，次の命令があります。

表 5.13 外部定義，外部参照疑似命令

疑似命令	概要
.public	オペランドに記述したシンボルをほかのモジュールから参照できるよう宣言
.extern	本モジュールで参照するほかのモジュールのビット・シンボル以外のシンボルを宣言

.public

オペランドに記述したシンボルをほかのモジュールから参照できるよう宣言します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.public	*ラベル名* [, *絶対式*]	[; *コメント*]

[機能]

- オペランドに記述したシンボルをほかのモジュールから参照できるよう宣言します。

[用途]

- ほかのモジュールから参照されるシンボルを定義している場合には，必ず，そのシンボルを .public 疑似命令で外部定義宣言します。

[詳細説明]

- 第 1 オペランドで指定したラベル名と同名のラベルを外部ラベル[注]として宣言します。
  なお，第 2 オペランドを指定した場合，指定した値をそのラベルの示すデータのサイズとして指定します。ただし，サイズの指定は無視されます (CX との互換性のため，記述は可能)。

  注　外部シンボル（GLOBAL のバインディング・クラスを持つシンボル）です。

- 本疑似命令と .extern 疑似命令は外部ラベルを宣言するという機能において変わりませんが，指定したファイル内に定義を持つラベルを外部ラベルとして宣言する場合は本疑似命令を用い，指定したファイル内に定義を持たないラベルを外部ラベルとして宣言する場合は .extern 疑似命令を用いるようにしてください。
- .public 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- .public 疑似命令は 1 行で 1 つのシンボルのみ定義可能です。
- オペランドに記述するシンボルが，同一モジュール内で定義されていない場合はワーニングを出力します。どのファイルにも定義がない場合，リンク時にエラーとなります。
- 次のシンボルは，オペランドとして記述することはできません。

  (a) .set 疑似命令で定義したシンボル

  (b) 外部参照シンボル（.extern 疑似命令で定義したシンボル）

  (c) セクション名

  (d) モジュール名

  (e) マクロ名

  (f) ソース・プログラム中で未定義なシンボル

[使用例]

- モジュール 1

```
        .public A1              ; (a)
        .extern B1

A1:
        .db2    0x10

        .cseg   text
        jr      B1
```

- モジュール 2

```
        .public B1              ; (b)
        .extern A1
        .cseg   text
B1:
        mov     A1, r12
```

(a) シンボル A1 が，ほかのモジュールから参照されるシンボルであることを宣言します。

(b) シンボル B1 が，ほかのモジュールから参照されるシンボルであることを宣言します。

.extern

本モジュールで参照するほかのモジュールのビット・シンボル以外のシンボルを宣言します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.extern	*ラベル名* [, *絶対式*]	[; *コメント*]

[機能]

- 本モジュールで参照するほかのモジュールのシンボルを宣言します。

[用途]

- ほかのモジュールの中で定義されているシンボルを参照する場合には，必ずそのシンボルを .extern 疑似命令で外部参照宣言します。

[詳細説明]

- 第 1 オペランドで指定したラベル名と同名のラベルを，外部ラベル[注]として宣言します。
  なお，第 2 オペランドを指定した場合，指定した値をそのラベルの示すデータのサイズとして指定します。ただし，サイズの指定は無視されます (CX との互換性のため，記述は可能)。

  注 外部シンボル（GLOBAL のバインディング・クラスを持つシンボル）です。

- 本疑似命令と .public 疑似命令は外部ラベルを宣言するという機能において変わりませんが，指定したファイル内に定義を持たないラベルを外部ラベルとして宣言する場合は本疑似命令を用い，指定したファイル内に定義を持つラベルを外部ラベルとして宣言する場合は .public 疑似命令を用いるようにしてください。
- .extern 疑似命令は，ソース・プログラムのどこに記述してもかまいません。
- .extern 疑似命令は 1 行で 1 つのシンボルのみ定義可能です。
- .extern 疑似命令で宣言されたシンボルをモジュール中で参照しなくても，エラーにはなりません。
- 第 1 オペランドの指定するシンボルが，本モジュールでの定義シンボル，.public 疑似命令にて宣言したシンボル，本疑似命令で宣言したシンボルであった場合，エラーとなります。その際，エラーになったシンボル名がメッセージ中に出力されます。
- 第 1 オペランドには，マクロ名を記述することはできません。

5.2.7 マクロ疑似命令

ソースを記述する場合，使用頻度の高い一連の命令群をそのつど記述するのは面倒です。また，記述ミス増加の原因ともなります。

マクロ疑似命令により，マクロ機能を使用することにより，同じような一連の命令群を何回も記述する必要がなくなり，コーディングの効率を上げることができます。

マクロの基本的な機能は，一連の文の置き換えにあります。

マクロ疑似命令には，次のものがあります。

表 5.14 マクロ疑似命令

疑似命令	概要
.macro	.macro 疑似命令と .endm 疑似命令の間に記述された一連の文に対し，シンボル欄で指定したマクロ名を付け，マクロを定義
.local	指定した文字列を特有の識別子として置き換えられるローカル・シンボルとして宣言
.rept	.rept 疑似命令と .endm 疑似命令の間に記述された一連の文をオペランド欄で指定した式の値分だけ，繰り返し展開
.irp	.irp 疑似命令と .endm 疑似命令の間にある一連の文をオペランドで指定された実パラメータで仮パラメータを置き換えながら，実パラメータの数だけ繰り返し展開
.exitm	.exitm 疑似命令を囲んでいる最も内側の .irp，.rept 疑似命令の繰り返しアセンブルをスキップ
.exitma	.exitma 疑似命令を囲んでいる最も外側の .irp，.rept 疑似命令の繰り返しアセンブルをスキップ
.endm	マクロの機能として定義される一連のステートメントを終了

.macro

.macro 疑似命令と .endm 疑似命令の間に記述された一連の文に対し，シンボル欄で指定したマクロ名を付け，マクロを定義します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
マクロ名	.macro : *マクロ・ボディ* : .endm	[*仮パラメータ* [, ...]]	[; *コメント*] [; *コメント*]

[機能]

- .macro 疑似命令と .endm 疑似命令の間に記述された一連の文（マクロ・ボディと呼びます）に対し，シンボル欄で指定したマクロ名を付け，マクロの定義を行います。

[用途]

- ソース中で，使用頻度の高い一連の文をマクロ定義しておきます。その定義以降では，定義されたマクロ名を記述するだけで，そのマクロ名に対応するマクロ・ボディが展開されます。

[詳細説明]

- .macro 疑似命令には，対応する .endm 疑似命令がなければ，エラーとなります。
- シンボル欄に記述するマクロ名の規則については，「(2) シンボル」を参照してください。
- マクロを参照する場合は，ニモニック欄に定義済みのマクロ名を記述します。
- オペランド欄に記述する仮パラメータの規則については，シンボル記述上の規則と同じです。
- 仮パラメータは，マクロ・ボディ内でのみ有効です。
- 仮パラメータとして予約語を記述すると，エラーとなります。ただし，ユーザ定義シンボルを記述した場合には，仮パラメータとしての認識が優先されます。
- 仮パラメータと実パラメータの個数は同じでなければなりません。実パラメータが足りない場合はエラーが出力されます。
- 仮パラメータの最大数は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。
- マクロ・ボディ内で定義したシンボル／ラベルを .local 疑似命令で宣言すれば，そのシンボル／ラベルは 1 回のマクロ展開でのみ有効になります。
- 1 つのモジュール内でのマクロ定義の最大数には，特に制限はありません。メモリが使えるかぎり定義することができます。
- クロスリファレンス・リストには，仮パラメータの定義行，参照行，シンボル名は出力されません。
- マクロ・ボディ中に，2 つ以上のセクションを定義することはできません。定義した場合は，エラーが出力されます。
- マクロ・ボディ内で参照されていない余分な仮パラメータがあった場合はエラーが出力されます。
- マクロ・ボディ内で，未定義なマクロの呼び出しが行われた場合，メッセージが出力され，アセンブルが中止されます。
- マクロ・ボディ内で，現在定義中のマクロの呼び出しが行われた場合，メッセージが出力され，アセンブルが中止されます。

- 仮パラメータにラベルや疑似命令で定義済みのパラメータを指定した場合，メッセージが出力され，アセンブルが中止されます。
- マクロ呼び出しにおいて実パラメータに指定可能なものは，ラベル名，シンボル名，数値，レジスタ，および命令ニモニックのみです。
  ラベル式（LABEL-1），参照方法指定ラベル（#LABEL），またはベース・レジスタ指定（[gp]）などを指定した場合，指定された実パラメータに依存したメッセージが出力され，アセンブルが中止されます。
- マクロ・ボディ内には，文の並びが指定可能です。オペランドなど，文の一部を指定することはできません。オペランドにマクロ呼び出しが存在する場合，マクロ名の未定義なラベル参照として扱われるか，メッセージが出力され，アセンブルが中止されます。
- マクロ定義中のマクロ・ボディ内で，マクロ定義を記述した場合は，エラーを出力し，処理が継続されます（対応する .endm 疑似命令までに記述された内容は無視されます）。そのマクロ名を参照した場合は定義エラーとなります。

［使用例］

```
ADMAC   .macro  PARA1, PARA2    ; (1)
        mov     PARA1, r12
        add     PARA2, r12
        .endm                   ; (2)

        ADMAC   0x10, 0x20      ; (3)
```

(1) マクロ名 “ADMAC”，2 つの仮引数 “PARA1”，“PARA2” を指定したマクロ定義をしています。

(2) マクロ定義の終わりを示します。

(3) マクロ ADMAC を参照しています。

.local

指定した文字列を特有の識別子として置き換えられるローカル・シンボルとして宣言します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.local	*シンボル名* [, ...]	[; *コメント*]

[機能]

- 指定したシンボル名をアセンブラ特有のシンボルとして置き換えられるローカル・シンボルとして宣言します。

[用途]

- マクロ・ボディ内でシンボルを定義しているマクロを 2 回以上参照すると，シンボルは二重定義エラーとなります。
  .local 疑似命令を使用することにより，シンボルを定義しているマクロを複数回，参照することができます。

[詳細説明]

- シンボル名の最大数は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。
- .local 疑似命令の仮パラメータに 4294967294 個以上のローカル・シンボルを指定した場合，次のメッセージが出力され，アセンブルが中止されます。

```
F0550514 ： 実パラメータが 4294967294 個以上用いられています。
```

- アセンブラによって生成されるローカル・シンボル名は，.??00000000 から .??FFFFFFFF までの範囲で生成されます。
- ローカル・シンボルとして宣言できるのは，未定義，または既にローカル・シンボル宣言しているシンボルのみです。
- 宣言ごとに，指定したシンボル名に対し置き換えられる特有なシンボルを生成します。
- 宣言されたシンボルを有効にするには，シンボルをラベルとして定義する必要があります。
- ローカル・シンボルとして宣言されたシンボル名は，ラベル以外のシンボルとして定義できません。

[使用例]

```
m1      .macro  x
        .local  a, b
        a:      .dw     a
        b:      .dw     x
.endm
m1      10
m1      20
```

展開すると次のようになります。

```
.??00000000:    .dw     .??00000000
.??00000001:    .dw     10
.??00000002:    .dw     .??00000002
.??00000003:    .dw     20
```

.rept

.rept 疑似命令と .endm 疑似命令の間に記述された一連の文をオペランド欄で指定した式の値分だけ，アセンブラが繰り返し展開します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.rept : .endm	*絶対式*	[; *コメント*] [; *コメント*]

[機能]

- .rept 疑似命令と .endm 疑似命令の間に記述された一連の文（REPT-ENDM ブロックと呼びます）をオペランド欄で指定した式の値分だけ，アセンブラが繰り返し展開します。

[用途]

- ソース中で一連の文を連続して繰り返し記述する場合に，.rept，.endm 疑似命令を使用します。

[詳細説明]

- .rept 疑似命令に対応する .endm 疑似命令がなければ，エラーとなります。
- REPT-ENDM のブロックの途中で .exitm が現れると，展開を中止します。
- REPT-ENDM のブロック内に，アセンブル制御命令を記述することができます。
- REPT-ENDM のブロック内に，マクロ定義を記述することはできません。
- 値は，32 ビットの符号付き整数として評価されます。
- 文の並び（ブロック）がない場合，何も行いません。
- 式の評価結果が負になった場合は，メッセージが出力され，アセンブルが中止されます。
- マクロ定義中のマクロ・ボディ内でマクロ定義を記述した場合は，エラーを出力し，展開処理を行わずに処理が継続されます。

[使用例]

```
.cseg   text
        ; REPT-ENDM ブロック
.rept   3               ; (1)
        nop
        ; ソース本文
.endm                   ; (2)
```

(1) REPT-ENDM ブロックを 3 回連続して展開するよう，指示しています。

(2) REPT-ENDM ブロックの終了を示します。

.irp

.irp 疑似命令と .endm 疑似命令の間にある一連の文をオペランドで指定された実パラメータ（左から順）で仮パラメータを置き換えながら，実パラメータの数だけ繰り返し展開します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.irp : .endm	*仮パラメータ* [*実パラメータ* [, ...]]	[; *コメント*] [; *コメント*]

[機能]

- .irp 疑似命令と .endm 疑似命令の間にある一連の文（IRP-ENDM ブロックと呼びます）をオペランドで指定された実パラメータ（左から順）で仮パラメータを置き換えながら，実パラメータの数だけ繰り返し展開します。

[用途]

- ソース中で，一部分だけ変数となる一連の文を連続して繰り返し記述したい場合に，IRP-ENDM 疑似命令を使用します。

[詳細説明]

- .irp 疑似命令には対応する .endm 疑似命令がなければ，エラーとなります。
- IRP-ENDM ブロックの途中に .exitm を記述すると，そこで展開を中止します。
- IRP-ENDM ブロックで，マクロを定義することはできません。
- IRP-ENDM ブロック内に，アセンブル制御命令を記述することができます。
- 実パラメータの最大数は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。
- 仮パラメータと実パラメータに同じパラメータ名を指定した場合，メッセージが出力され，アセンブルが中止されます。
- 仮パラメータと実パラメータにラベルや他の疑似命令で定義済みのパラメータ名を指定した場合，メッセージが出力され，アセンブルが中止されます。
- マクロ定義中のマクロ・ボディ内でマクロ定義を記述した場合は，エラーを出力し，展開処理を行わずに処理が継続されます。

[使用例]

```
.cseg   text

.irp    PARA 0xA, 0xB, 0xC      ; (1)
    ; IRP-ENDM ブロック
        add     PARA, r12
        mov     r11, r12
.endm                           ; (2)
    ; ソース本文
```

(1) 仮パラメータが PARA，実パラメータが 0xA，0xB，0xC の 3 個です。
仮パラメータ “PARA” を実引数 “0xA”，“0xB”，“0xC” に置き換えながら，IRP-ENDM ブロックを実パラメータの数 3 回分展開することを指示します。

(2) IRP-ENDM ブロックの終了を示します。

.exitm

本疑似命令を囲んでいる最も内側の .irp，.rept 疑似命令の繰り返しアセンブルをスキップします。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.exitm		[; *コメント*]

[機能]

- 本疑似命令は，本疑似命令を囲んでいる最も内側の .irp，.rept 疑似命令の繰り返しアセンブルをスキップします。

[詳細説明]

- 本疑似命令が .irp，.rept 疑似命令に囲まれていない場合，メッセージが出力され，アセンブルが中止されます。

.exitma

本疑似命令は，本疑似命令を囲んでいる最も外側の .irp，.rept 疑似命令の繰り返しをスキップします。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
[*ラベル* :]	.exitma		[; *コメント*]

[機能]

- 本疑似命令は，本疑似命令を囲んでいる最も外側の .irp，.rept 疑似命令の繰り返しをスキップします。

[詳細説明]

- 本疑似命令が .irp，.rept 疑似命令に囲まれていない場合，メッセージが出力され，アセンブルが中止されます。

.endm

マクロの機能として定義される一連のステートメントの終了をアセンブラに指示します。

[指定形式]

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.endm		[; コメント]

[機能]

- マクロの機能として定義される一連のステートメントの終了をアセンブラに指示します。

[用途]

- .macro 疑似命令，.rept 疑似命令，および .irp 疑似命令に続く一連のマクロ・ステートメントの最後には，必ず .endm 疑似命令を記述します。

[詳細説明]

- .macro 疑似命令と .endm 疑似命令の間に記述された一連のマクロ・ステートメントがマクロ・ボディとなります。
- .rept 疑似命令と .endm 疑似命令の間に記述された一連のステートメントが，REPT-ENDM ブロックとなります。
- .irp 疑似命令と .endm 疑似命令の間に記述された一連のステートメントが，IRP-ENDM ブロックとなります。
- 本疑似命令に対応する .macro 疑似命令，.rept 疑似命令，および .irp 疑似命令が存在しない場合，メッセージが出力され，アセンブルが中止されます。

[使用例]

（1）MACRO-ENDM

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

(3) REPT-ENDM

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

(4) IRP-ENDM

```
.cseg   text
  :
.irp    PARA, <1, 2, 3>
        add     A, #PARA
        mov     [DE], A
.endm
```

5.3 制御命令

制御命令とは，アセンブラの動作に対し細かい指示を与えるものです。

5.3.1 概　　要

制御命令は，アセンブラの動作に対し細かい指示を与えるもので，ソース中に記述します。
制御命令は，機械語生成の対象とはなりません。
次に，制御命令の種類を示します。

表 5.15　　制御命令一覧

制御命令の種類	制御命令
コンパイル対象品種指定制御命令	PROCESSOR
アセンブラ制御命令	REG_MODE，NOMACRO，MACRO，DATA，SDATA，NOWARNING，WARNING
ファイル入力制御命令	INCLUDE，BINCLUDE
条件アセンブル制御命令	IFDEF，IFNDEF，IF，IFN，ELSEIF，ELSEIFN，ELSE，ENDIF

制御命令は，疑似命令と同様に，ソース中に記述します。

5.3.2 アセンブラ制御命令

アセンブラ制御命令を用いることにより，アセンブラが行う処理を制御できます。
アセンブラ制御命令には，次のものがあります。

表 5.16 アセンブラ制御命令

制御命令	概要
REG_MODE	レジスタ・モード情報セクションを出力
NOMACRO	命令展開の抑制
MACRO	NOMACRO による指定を解除
DATA	シンボル名の外部データが data 再配置属性，または bss 再配置属性のセクションに割り当てられているとみなした命令展開
SDATA	シンボル名の外部データが sdata 再配置属性，または sbss 再配置属性のセクションに割り当てられているとみなした命令展開の抑制
NOWARNING	警告メッセージの出力を抑止
WARNING	警告メッセージを出力

REG_MODE

レジスタ・モード情報セクションを出力します。

[指定形式]

```
[ △ ]$[ △ ]REG_MODE[ △ ] レジスタ・モード指定 [ △ ][; コメント ]
```

[機能]

- アセンブラが生成するオブジェクト・ファイル中に，レジスタ・モード情報セクションを出力します。

[詳細説明]

- レジスタ・モード指定には，レジスタ・モード 22 を示す “22”，レジスタ・モード 32 を示す “32”，共通レジスタ・モードを示す “common” のいずれかを指定します。
- レジスタ・モード情報セクションとは，コンパイラが使用する作業用レジスタとレジスタ変数用レジスタの本数情報を保持するもので，本制御命令によりオブジェクト・ファイルに設定されます。
- レジスタ・モード 22 を使用する場合，作業用レジスタとレジスタ変数用レジスタはそれぞれ 5 本ずつ，レジスタ・モード 32 を使用する場合はそれぞれ 10 本ずつとなります。
- レジスタ・モード 32 を使用する場合は，アセンブラが生成するオブジェクト・ファイル中に，レジスタ・モード情報セクションを出力しません。
- 本制御命令でのレジスタ・モード指定と，オプションによるレジスタ・モード指定が異なる場合，CC-RH はワーニングを出力し，オプションによるレジスタ・モード指定を無視します。
- 本制御命令でのレジスタ・モード指定が複数行あり，それらのレジスタ・モード指定が異なる場合，最初のレジスタ・モード指定が有効となります。また，異なるレジスタ・モード指定に対して，CC-RH はワーニングを出力してレジスタ・モード指定を無視します。

NOMACRO

命令展開を行いません。

[指定形式]

```
[ △ ]$[ △ ]NOMACRO[ △ ][; コメント]
```

[機能]

- それ以降の命令に対し，命令展開を行いません。

MACRO

NOMACRO による指定を解除します。

[指定形式]

```
[ △ ]$[ △ ]MACRO[ △ ][; コメント ]
```

[機能]

- それ以降の命令に対し，NOMACRO による指定を解除します。

DATA

シンボル名の外部データが data 再配置属性，または bss 再配置属性のセクションに割り当てられているとみなし，命令展開を行います。

[指定形式]

```
[ △ ]$[ △ ]DATA[ △ ] シンボル名 [ △ ][; コメント ]
```

[機能]

- シンボル名の外部データが，data 再配置属性，または bss 再配置属性のセクションに割り当てられているとみなし，そのデータを参照する命令に対して命令展開を行います。
- #pragma section で “data” を指定した変数を，アセンブリ言語ソース・ファイルで外部参照する場合に利用します。

SDATA

シンボル名の外部データが sdata 再配置属性，または sbss 再配置属性のセクションに割り当てられているとみなし，命令展開を行いません。

［指定形式］

```
[ △ ]$[ △ ]SDATA[ △ ] シンボル名 [ △ ][; コメント ]
```

［機能］

- シンボル名の外部データが，sdata 再配置属性，または sbss 再配置属性のセクションに割り当てられているとみなし，そのデータを参照する命令に対して命令展開を行いません。
- #pragma section で “sdata” を指定した変数を，アセンブリ言語ソース・ファイルで外部参照する場合に利用します。

NOWARNING

警告メッセージの出力を抑止します。

[指定形式]

```
[△]$[△]NOWARNING[△][;コメント]
```

[機能]

- それ以降の命令に対し，警告メッセージの出力を抑止します。

WARNING

警告メッセージを出力します。

[指定形式]

```
[ △ ]$[ △ ]WARNING[ △ ][; コメント]
```

[機能]

- それ以降の命令に対し，警告メッセージを出力します。

5.3.3 ファイル入力制御命令

ファイル入力制御命令を用いることにより，アセンブリ・ソース・ファイル，またはバイナリ・ファイルを，指定した位置に取り込むことができます。
ファイル入力制御命令には，次のものがあります。

表 5.17 ファイル入力制御命令

制御命令	概要
INCLUDE	ほかのソース・モジュール・ファイルの一連のステートメントを引用
BINCLUDE	バイナリ・ファイルの入力

INCLUDE

ほかのソース・モジュール・ファイルの一連のステートメントを引用します。

[指定形式]

```
[ △ ]$[ △ ]INCLUDE[ △ ]([ △ ] ファイル名 [ △ ])[ △ ][; コメント ]
```

[機能]

- 指定されたファイルの内容を指定された行以降に挿入展開し，アセンブルします。

[用途]

- 複数のソース・モジュール中で共通に記述する比較的大きな一連のステートメントを 1 つのファイル（インクルード・ファイル）としてまとめておきます。
  各ソース・モジュール中で，その一連のステートメントを引用する必要があるとき，INCLUDE 制御命令により，必要とするインクルード・ファイル名を指定します。
  これにより，ソース・モジュールの記述作業を軽減することができます。

[詳細説明]

- INCLUDE 制御命令は，通常のソースにのみ記述することができます。
- オプション（-I）で，インクルード・ファイルのサーチ・パスを指定することができます。
- インクルード・ファイルの読み込みパスのサーチの順番は，次のとおりです。
  (a) オプション（-I）で指定されたフォルダ
  (b) 標準インクルード・ファイル・フォルダ
  (c) ソース・ファイルのあるフォルダ
  (d) （元の）C ソース・ファイルのあるフォルダ
  (e) カレント・フォルダ
- インクルード・ファイルは，ネスティングすることが可能です（ネスティングとは，インクルード・ファイル中で，別のインクルード・ファイルを指定することです）。
- インクルード・ファイルのネスト・レベルの最大値は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。
- インクルード・ファイルがオープンできない場合，メッセージが出力され，アセンブルが中止されます。
- 1 つのインクルード・ファイル中に，セクション定義疑似命令やマクロ定義疑似命令や条件アセンブル制御命令のような開始から終了までのブロックがあるものは，その対応が取れた状態で，閉じなければなりません。対応が取れてないあるいは閉じていない場合はエラーが出力され，閉じられたものとしてアセンブルを継続します。
- アセンブル対象とならないセクション定義疑似命令やマクロ定義疑似命令や条件アセンブル制御命令についてはチェックを行いません。

BINCLUDE

バイナリ・ファイルの入力をします。

[指定形式]

```
[ △ ]$[ △ ]BINCLUDE[ △ ]([ △ ] ファイル名 [ △ ])[ △ ][; コメント ]
```

[機能]

- オペランドに指定したバイナリ・ファイルの内容を，本制御命令の置かれている位置に置かれたソース・プログラムのアセンブル結果であるとみなして扱います。

[詳細説明]

- オプション（-I）で，インクルード・ファイルのサーチ・パスを指定することができます。
- インクルード・ファイルの読み込みパスのサーチの順番は，次のとおりです。
  (a) オプション（-I）で指定されたフォルダ
  (b) 標準インクルード・ファイル・フォルダ
  (c) ソース・ファイルのあるフォルダ
  (d) （元の）C ソース・ファイルのあるフォルダ
  (e) カレント・フォルダ
- 本制御命令は，バイナリ・ファイルの内容全体を扱います。リロケータブル・ファイルを指定した場合には，ELF フォーマットで構成されたファイル全体を扱います。.text セクション等の内容のみを扱うということではありません。
- 存在しないファイルを指定した場合，メッセージが出力され，アセンブルが中止されます。

5.3.4 条件アセンブル制御命令

条件アセンブル制御命令を用いることにより，条件式の評価結果にしたがって，アセンブルを行う範囲が制御できます。

条件アセンブル制御命令には，次のものがあります。

表 5.18 条件アセンブル疑似命令

疑似命令	意味
IFDEF	シンボルによる制御（定義されているときアセンブル）
IFNDEF	シンボルによる制御（定義されていないときアセンブル）
IF	絶対値式による制御（真のときアセンブル）
IFN	絶対値式による制御（偽のときアセンブル）
ELSEIF	絶対値式による制御（真のときアセンブル）
ELSEIFN	絶対値式による制御（偽のときアセンブル）
ELSE	絶対値式／シンボルによる制御
ENDIF	制御範囲の終わり

条件アセンブル制御命令のネスト・レベルの最大値は 4,294,967,294（=0xFFFFFFFE）（理論値）です。ただし，実際には利用可能なメモリ量に依存します。

IFDEF

シンボルによる制御（定義されているときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IFDEF[ △ ] スイッチ名 [ △ ][; コメント ]
```

[機能]

- オペランドで指定したスイッチ名が定義されている場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令とその制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

- 指定したスイッチ名が定義されていない場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- スイッチ名記述上の規則は，シンボル記述上の規則（「(2) シンボル」を参照してください）と同じです。
- スイッチ名は，予約語以外のユーザ定義シンボルと重複してもかまいません。ただし，スイッチ名同士の重複チェックは行われます。
- スイッチ名は，アセンブル・リスト・ファイルのシンボル・リスト情報，およびクロスリファレンス情報には出力されません。

IFNDEF

シンボルによる制御（定義されていないときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IFNDEF[ △ ] スイッチ名 [ △ ][; コメント ]
```

[機能]

- オペランドで指定したスイッチ名が定義されている場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

- 指定したスイッチ名が定義されていない場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令と本制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- スイッチ名記述上の規則は，シンボル記述上の規則（「(2) シンボル」を参照してください）と同じです。
- スイッチ名は，予約語以外のユーザ定義シンボルと重複してもかまいません。ただし，スイッチ名同士の重複チェックは行われます。
- スイッチ名は，アセンブル・リスト・ファイルのシンボル・リスト情報，およびクロスリファレンス情報には出力されません。

IF

絶対値式による制御（真のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IF[ △ ] 絶対値式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対値式が真（≠ 0）に評価された場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令と本制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

- 偽（=0）に評価された場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- 絶対式の値は 32 ビットの符号付き整数として評価されます。

IFN

絶対値式による制御（偽のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IFN[ △ ] 絶対値式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対値式が真（≠ 0）に評価された場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

- 偽（=0）に評価された場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令と本制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- 絶対式の値は 32 ビットの符号付き整数として評価されます。

ELSEIF

絶対値式による制御（真のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]ELSEIF[ △ ] 絶対値式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対値式が真（≠ 0）に評価された場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令とその制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

- 偽（=0）に評価された場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- 絶対式の値は 32 ビットの符号付き整数として評価されます。

ELSEIFN

絶対値式による制御（偽のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]ELSEIFN[ △ ] 絶対値式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対値式が真（≠ 0）に評価された場合
  本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，ELSE 制御命令，または ENDIF 制御命令までスキップします。

- 偽（=0）に評価された場合

  (a) 本制御命令と本制御命令に対応する ELSEIF 制御命令，ELSEIFN 制御命令，または ELSE 制御命令が存在する場合は，本制御命令とその制御命令とで囲まれるブロックをアセンブルします。

  (b) それらの制御命令が存在しない場合は，本制御命令とその制御命令に対応する ENDIF 制御命令とで囲まれるブロックをアセンブルします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。
- 絶対式の値は 32 ビットの符号付き整数として評価されます。

ELSE

絶対値式／シンボルによる制御をします。

[指定形式]

[△]$[△]ELSE[△] *絶対値式* [△][; *コメント*]

[機能]

- IFDEF 制御命令においてスイッチ名が定義されていない場合，IF 制御命令，または ELSEIF 制御命令において絶対値式が偽（=0）に評価された場合，あるいは IFN 制御命令，ELSEIFN 制御命令において絶対値式が真（≠ 0）に評価された場合，本制御命令と本制御命令に対応する ENDIF 制御命令 . とで囲まれる文の並び（ブロック）をアセンブルします。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。

ENDIF

制御範囲の終わりを示します。

[指定形式]

```
[ △ ]$[ △ ]ENDIF[ △ ] 絶対値式 [ △ ][; コメント ]
```

[機能]

条件アセンブル制御命令による制御の範囲の終わりを示します。

[用途]

- ソース・モジュールを大幅に変更することなく，アセンブル対象となるソース・ステートメントを変更することができます。
- ソース・モジュール中に，プログラム開発中にのみ必要となるデバッグ文などを記述した場合，そのデバッグ文を機械語に変換する／しないを条件付きアセンブルのスイッチ設定により選択することができます。

[詳細説明]

- 本制御命令は，通常のソース・プログラム中に記述することができます。

5.4 マクロ

この節では，マクロ機能の使い方について説明します。
プログラムの中で一連の命令群を何回も記述する場合に使用すると，便利な機能です。

5.4.1 概　　要

ソースの中で一連の命令群を何回も記述する場合，マクロ機能を使用すると便利です。
マクロ機能とは，.macro，.endm 疑似命令により，マクロ・ボディとして定義された一連の命令群をマクロ参照している箇所に展開することです。
マクロは，ソースの記述性を向上させるために使用するもので，サブルーチンとは異なります。
マクロとサブルーチンには，それぞれ次のような特徴があります。それぞれ目的に応じて有効に使用してください。

- サブルーチン
  プログラム中で何回も必要となる処理を 1 つのサブルーチンとして記述します。サブルーチンは，アセンブラにより一度だけ機械語に変換されます。
  サブルーチンの参照には，サブルーチン・コール命令（一般にはその前後に引数設定の命令が必要）を記述するだけで済みます。したがって，サブルーチンを活用することにより，プログラムのメモリを効率よく使用することができます。
  プログラム中の一連のまとまった処理をサブルーチン化することにより，プログラムの構造化を図ることができます（プログラムを構造化することにより，プログラム全体の構造が分かりやすくなり，プログラムの設計が容易になります）。

- マクロ
  マクロの基本的な機能は，命令群の置き換えです。
  .macro，.endm 疑似命令によりマクロ・ボディとして定義された一連の命令群が，マクロ参照時にその場所に展開されます。アセンブラは，マクロ参照を検出するとマクロ・ボディを展開し，マクロ・ボディの仮パラメータを参照時の実パラメータに置き換えながら，命令群を機械語に変換します。
  マクロは，パラメータを記述することができます。
  たとえば，処理手順は同じであるがオペランドに記述するデータだけが異なる命令群がある場合，そのデータに仮パラメータを割り当ててマクロを定義します。マクロ参照時には，マクロ名と実パラメータを記述することにより，記述の一部分だけが異なる種々の命令群に対処することができます。

サブルーチン化の手法が，メモリ・サイズの削減やプログラムの構造化を図るために用いられるのに対し，マクロは，コーディングの効率を向上させるために用いられます。

5.4.2 マクロの利用

マクロとは，一連の決まった手順パターンを登録し，それを利用して記述するものです。マクロはユーザが定義します。マクロの定義方法は次のように，マクロ本体を “.macro” と “.endm”，で囲む形になります。

```
PUSHMAC .macro  REG             ; 次の 2 つの文がマクロ本体
        add     -4, sp
        st.w    REG, 0x0[sp]
.endm
```

上記を定義したあと，次のように記述した場合，「r19 をスタックに格納する」というコードに置き換えられます。

```
PUSHMAC r19
```

したがって，次のようなコードに展開されます。

```
add     -4, sp
st.w    r19, 0x0[sp]
```

5.4.3 マクロ・オペレータ

この項では，マクロ本体内において文字列と文字列を連結するコンカティネート記号 “~”，およびダラー記号 “$” について説明します。

(1) ~（コンカティネート）

- コンカティネート記号は，マクロ・ボディ内で文字，または文字列と文字，または文字列を連結します。マクロ展開時には，コンカティネート記号の左右の文字，または文字列を連結し，コンカティネート自身は消滅します。
- コンカティネート記号は，マクロ定義時にシンボル中の “~” の前後を仮パラメータ，あるいはローカル・シンボルとして認識することが可能であり，区切り記号として利用することもできます。マクロ展開時にシンボル中の “~” の前後の仮パラメータ，あるいはローカル・シンボルを評価してシンボル中に連結することができます。
- コンカティネート記号としての “~” は，マクロ定義時のみ有効です。
- 文字列中，およびコメント中の “~” は，単なるデータとして扱われます。
- “~” を 2 つ続けると，単なるデータとして扱われます。

例 1.

```
abc     .macro  x
        abc~x:  mov     r10, r20
                sub     def~x, r20
.endm
abc STU
```

【展開結果】

```
abcSTU: mov     r10, r20
        sub     defSTU, r20
```

例 2.

```
abc     .macro  x, xy
        a_~xy:  mov     r10, r20
        a_~x~y: mov     r20, r10
.endm
abc stu, STU
```

【展開結果】

```
a_STU:  mov     r10, r20
a_stuy: mov     r20, r10
```

例 3.

```
abc     .macro  x, xy
        ~ab:    mov r10, r20
.endm
abc stu, STU
```

【展開結果】

```
ab:     mov     r10, r20
```

(2) $（ダラー記号）

アセンブラでは，マクロ呼び出しにおける実引数としてダラー記号 “$” を前に置いたシンボルを指定した場合，そのシンボルの値が実引数として指定されたものみなされます。ただし，ダラー記号 $ に続けてシンボル以外の識別名，または未定義シンボル名を指定した場合，メッセージが出力され，アセンブルが中止されます。

例

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

【展開結果】

```
.set    value, 10
mov     value, r10
mov     10, r10
```

5.5 予 約 語

アセンブラには予約語が存在します。予約語をシンボル，ラベル，セクション名，マクロ名に使用することはできません。予約語を指定した場合は，メッセージが出力され，アセンブルが中止されます。予約語は大文字／小文字を区別しません。

予約語は次のとおりです。

- 命令（add，sub，mov など）
- 疑似命令
- 制御命令
- レジスタ名，内部レジスタ名
- GHS の予約セクション（“_GHS” と “.ghs”，および “.__ghs” で始まるセクション名）

5.6 あらかじめ定義されたマクロ名

アセンブラでは以下のマクロを定義します。

定義済みマクロ名	備考
RENESAS	値は 1
ASRH	値は 1
_ASRH	値は 1
_RH850_	値は 1
_RH850	値は 1

5.7 アセンブラ生成シンボル

次に，アセンブラが内部処理で利用するために生成するシンボルの一覧を示します。
以下のシンボルと同名のシンボルは利用できません。
アセンブラは，“.” で始まるシンボルを内部処理用のシンボルとして，オブジェクト・ファイルへは出力しません。ただし，予約セクション名は除きます。

表 5.19 アセンブラ生成シンボル

シンボル名	説明
.??00000000 ～ .??FFFFFFFF	.local 疑似命令生成ローカル・シンボル

シンボル名	説明
.LMn_n （n：0 ～ 4294967294 ）	例 .LM0_1

5.8 命令セット

この節では，CC-RH がサポートする命令セットについて説明します。

(1) 記号の説明
次表に，以降で用いる記号の意味を示します。

表 5.20 記号の意味

記号	意味
CMD	命令
CMDi	命令（andi，ori，または xori）
reg，reg1，reg2	レジスタ
r0，R0	ゼロ・レジスタ
R1	アセンブラ予約レジスタ（r1）
gp	グローバル・ポインタ（r4）
ep	エレメント・ポインタ（r30）
[reg]	ベース・レジスタ
disp	ディスプレースメント（アドレスからの偏位） 特に記述のない場合 32 ビット幅を持ちます。
disp*n*	*n* ビットのディスプレースメント
imm	イミーディエト（即値） 特に記述のない場合 32 ビット幅を持ちます。
imm*n*	*n* ビットのイミーディエト
bit#3	ビット・ナンバ指定用 3 ビット・データ
cc#3	浮動小数点システム・レジスタ FPSR の CC0 ～ CC7(24 ～ 31 ビット) 指定用 3 ビット・データ
#label	ラベルの絶対アドレス参照
label	ラベルのセクション内オフセット参照，または PC オフセット参照
$label	ラベルの gp オフセット参照
!label	ラベルの絶対アドレス参照（命令展開なし）
%label	ラベルのセクション内オフセット参照（命令展開なし），または ep オフセット参照
HIGHW(*value*)	*value* の上位 16 ビット
LOWW(*value*)	*value* の下位 16 ビット
HIGHW1(*value*)	*value* の上位 16 ビット ＋ *value* のビット番号 15 のビット値[注]
HIGH(*value*)	*value* の下位 16 ビット中の上位 8 ビット
LOW(*value*)	*value* の下位 8 ビット
addr	アドレス
PC	プログラム・カウンタ
PSW	プログラム・ステータス・ワード
regID	システム・レジスタ番号（0 ～ 31）
vector	トラップ・ベクタ（0 ～ 31）

記号	意味
BITIO	周辺 I/O レジスタ（1 ビット操作専用）

注 LSB（Least Significant Bit）はビット番号 0 です。

(2) オペランド

以下に，アセンブラにおけるオペランドの記述形式について説明します。アセンブラでは，命令，および疑似命令に対するオペランドとして，レジスタ，定数，シンボル，ラベル参照，および定数，シンボル，ラベル参照，演算子，かっこで構成した式を指定できます。

(a) レジスタ

アセンブラにおいて指定できるレジスタを次に示します。注

r0，zero，r1，r2，hp，r3，sp，r4，gp，r5，tp，r6，r7，r8，r9，r10，r11，r12，r13，r14，r15，r16，r17，r18，r19，r20，r21，r22，r23，r24，r25，r26，r27，r28，r29，r30，ep，r31，lp

注 PSW，およびシステム・レジスタは，ldsr/stsr 命令において，番号で指定します。なお，アセンブラでは，PC をオペランドに指定する方法はありません。

r0 と zero（ゼロ・レジスタ），r2 と hp（ハンドラ・スタック・ポインタ），r3 と sp（スタック・ポインタ），r4 と gp（グローバル・ポインタ），r5 と tp（テキスト・ポインタ），r30 と ep（エレメント・ポインタ），r31 と lp（リンク・ポインタ）は同じレジスタを示します。

(b) r0

r0 は，常に 0 の値を持つレジスタです。したがって，デスティネーション・レジスタとして指定した場合にも，結果の代入は行われません。なお，r0 をデスティネーション・レジスタとして指定した場合，次のメッセージが出力され注，アセンブルが続行されます。

注 このメッセージの出力は，アセンブラの起動時に警告メッセージ抑止オプション（-Xno_warning）を指定することにより抑止できます。

```
mov     0x10, r0
    ↓
W0550013 : register r1 が、destination レジスタとしてオペランドに指定されています
```

(c) r1

アセンブラ予約レジスタ（r1）は，アセンブラにおいて，命令展開を行う際のテンポラリ・レジスタとして用いられるレジスタです。なお，r1 をソース・レジスタ，またはデスティネーション・レジスタとして指定した場合，次のメッセージが出力され注，アセンブルが続行されます。

注 このメッセージの出力は，アセンブラの起動時に警告メッセージ抑止オプション（-Xno_warning）を指定することにより抑止できます。

```
mov     0x10, r1
    ↓
W0550013 : register r1 が、destination レジスタとしてオペランドに指定されています。
```

```
mov     r1, r10
    ↓
W0550013 : register r1 が、source レジスタとしてオペランドに指定されています。
```

(d) 定数

アセンブラでは，命令，および疑似命令のオペランド指定で使用可能な絶対値式，または相対値式の構成要素として，整定数，および文字定数を用いることができます。
また，.float 疑似命令のオペランド指定には浮動小数点定数を用いることができます。

(e) シンボル

アセンブラでは，命令，および疑似命令のオペランド指定で使用可能な絶対値式，または相対値式の構成要素として，シンボルを用いることができます。

(f) ラベル参照

アセンブラでは，次に示した命令／疑似命令のオペランド指定で，使用可能な相対値式の構成要素として，ラベル参照を用いることができます。

- メモリ参照命令（ロード／ストア命令，およびビット操作命令）
- 演算命令（算術演算命令，飽和演算命令，および論理演算命令）
- 分岐命令
- 領域確保疑似命令

アセンブラでは，ラベル参照は参照方法の違い，およびそのラベル参照を用いている命令／疑似命令の違いにより次に示すように異なる意味を持ちます。

表 5.21 ラベル参照

参照方法	用いている命令	意味
#label	メモリ参照命令，演算命令，jmp 命令	ラベル label 定義の存在する位置の絶対アドレス（アドレス 0 からのオフセット注 1）。 32 ビットのアドレスを持ち，ld23，st23，mov 命令以外は，命令展開されます。
	領域確保疑似命令	ラベル label 定義の存在する位置の絶対アドレス（アドレス 0 からのオフセット注 1）。 ただし，32 ビットのアドレスを，確保した領域の大きさに準じてマスクした値。
!label	メモリ参照命令，演算命令	ラベル label 定義の存在する位置の絶対アドレス（アドレス 0 からのオフセット注 1）。 16 ビットのアドレスを持ち，16 ビット・ディスプレースメント，またはイミーディエトをもつ命令に指定した場合，命令展開は行われません。 その他の命令に指定した場合，命令展開されます。 ラベル label の定義したアドレスが，16 ビットで表現できる範囲でない場合，リンク時にエラーになります。
	領域確保疑似命令	ラベル label 定義の存在する位置の絶対アドレス（アドレス 0 からのオフセット注 1）。 ただし，32 ビットのアドレスを，確保した領域の大きさに準じてマスクした値。
label	メモリ参照命令，演算命令	ラベル label 定義の存在する位置のセクション内オフセット（ラベル label 定義の存在するセクションの先頭アドレスからのオフセット注 2）。 32 ビットのオフセットを持ち，ld23，st23，mov 命令以外は，命令展開されます。
	jmp 命令を除く分岐命令	ラベル label 定義の存在する位置の PC オフセット（ラベル label 参照を用いている命令の先頭アドレスからのオフセット）。
	領域確保疑似命令	ラベル label 定義の存在する位置のセクション内オフセット（ラベル label 定義の存在するセクションの先頭アドレスからのオフセット注 2）。 ただし，32 ビットのオフセットを，確保した領域の大きさに準じてマスクした値。

参照方法	用いている命令	意味
%label	メモリ参照命令，演算命令	16 ビットのオフセットを持ち，16 ビット・ディスプレースメント，またはイミーディエトをもつ命令に指定した場合，命令展開は行われません。 その他の命令に指定した場合，命令展開されます。 ラベル label の定義したアドレスが，16 ビットで表現できる範囲でない場合，リンク時にエラーになります。
	領域確保疑似命令	ラベル label 定義の存在する位置の ep オフセット (エレメント・ポインタの示すアドレスからのオフセット)。 ただし，32 ビットのオフセットを，確保した領域の大きさに準じてマスクした値。
$label	メモリ参照命令，演算命令	ラベル label 定義の存在する位置の gp オフセット（グローバル・ポインタの指すアドレスからのオフセット）。

注 1. リンク後のオブジェクト・ファイルにおけるアドレス 0 からのオフセットです。

注 2. リンク後のオブジェクト・ファイルにおいて，ラベル label 定義の存在するセクションが割り当てられたセクション（出力セクション）の先頭アドレスからのオフセットです。

次に，メモリ参照命令，演算命令，分岐命令，および領域確保疑似命令におけるラベル参照の意味を示します。

表 5.22 メモリ参照命令

参照方法	意味
#label[reg]	ラベル label の絶対アドレスがディスプレースメントとして扱われます。 32 ビットの値を持ち，ld23，st23 命令以外は，命令展開されます。#label[r0] とすることにより絶対アドレスによる参照を指定できます。 [reg] の部分が省略でき，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
label[reg]	ラベル label のセクション内オフセットがディスプレースメントして扱われます。32 ビットの値を持ち，ld23，st23 命令以外は，命令展開されます。reg に対象とするセクションの先頭アドレスを指すレジスタを指定し，label[reg] とすることにより，一般的なレジスタ相対の参照が指定できます。
$label[reg]	ラベル label の gp オフセットがディスプレースメントとして扱われます。ラベル label の定義されたセクションにより，32，または 16 ビットの値を持ち，命令展開のパターンが変化します注。16 ビットの値を持つ命令展開が行われた場合，ラベル label の定義したアドレスより算出されたオフセットが 16 ビットで表現できる範囲でない場合，リンク時にエラーになります。$label[gp] とすることにより gp レジスタ相対の参照（gp オフセット参照と呼ぶ）が指定できます。[reg] の部分が省略でき，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。
!label[reg]	ラベル label の絶対アドレスがディスプレースメントとして扱われます。16 ビットの値を持ち，命令展開は行われません。ラベル label に定義したアドレスが 16 ビットで表現できない場合，リンク時にエラーになります。!label[r0] とすることにより絶対アドレスによる参照が指定できます。 [reg] の部分が省略でき，省略した場合は [r0] が指定されたものとみなされます。 ただし，#label[reg] 参照とは異なり，命令展開は行われません。
%label[reg]	ラベル label 定義の存在する位置の ep シンボルからのオフセットがディスプレースメントとして扱われます。 16 ビット，または命令によってはそれ以下の値を持ち，その範囲で表現できる値でない場合，リンク時にエラーになります。 [reg] の部分が省略でき，省略した場合，アセンブラでは，[ep] が指定されたものとみなされます。

注 「(h) gp オフセット参照」を参照してください。

表 5.23 演算命令

参照方法	意味
#label	ラベル label の絶対アドレスがイミーディエトとして扱われます。 32 ビットの値を持ち，mov 命令以外は，命令展開されます。
label	ラベル label のセクション内オフセットがイミーディエトとして扱われます。 32 ビットの値を持ち，mov 命令以外は，命令展開されます。
$label	ラベル label の gp オフセットがイミーディエトとして扱われます。 ラベル label の定義されたセクションにより，32，または 16 ビットの値を持ち，命令のパターンが変化します[注 1]。16 ビットの値を持つ展開をされた場合，ラベル label の定義したアドレスより算出されたオフセットが 16 ビットで表現できる範囲でない場合，リンク時にエラーになります。
!label	ラベル label の絶対アドレスがイミーディエトとして扱われます。 16 ビットの値を持ち，イミーディエトとして 16 ビットの値を指定できるアーキテクチャの演算命令[注 2] に指定した場合，命令展開は行われません。add，mov，および mulh 命令に指定した場合，命令展開されます。それ以外の命令に指定することはできません。16 ビットで表現できる範囲でない場合，リンク時にエラーになります。
%label	ラベル label 定義の存在する位置の ep シンボルからのオフセットがイミーディエトとして扱われます。 16 ビットの値を持ち，イミーディエトとして 16 ビットの値を指定できるアーキテクチャの演算命令[注 2] に指定した場合，命令展開は行われません。 また，この参照方法は，イミーディエトとして 16 ビットの値を指定できるアーキテクチャの演算命令と，add，mov，および mulh 命令のみで指定できます。add，mov，および mulh 命令に指定した場合，命令展開されます。それ以外の命令に指定することはできません。16 ビットで表現できる範囲でない場合，リンク時にエラーになります

注 1. 「(h) gp オフセット参照」を参照してください。

注 2. イミーディエトとして 16 ビットの値を指定できる命令は，addi，andi，movea，mulhi，ori，satsubi，および xori です。

表 5.24 分岐命令

参照方法	意味
#label	jmp 命令において，ラベル label の絶対アドレスが飛び先アドレスとして扱われます。 32 ビットの値を持ち，命令展開されます。
label	jmp 命令以外の分岐命令において，ラベル label の PC オフセットがディスプレースメントとして扱われます。 22 ビットの値を持ち，表現できない範囲である場合，リンク時にエラーになります。

表 5.25 領域確保疑似命令

参照方法	意味
#label !label	.db4/.db2/.db 疑似命令において，ラベル label の絶対アドレスを値として扱われます。 32 ビットの値を持ちますが，各疑似命令のビット幅に応じてマスクされます。
label	.db4/.db2/.db 疑似命令において，ラベル label 定義の定義されたセクション内オフセットを値として扱われます。 32 ビットの値を持ちますが，各疑似命令のビット幅に応じてマスクされます。
%label	.db4/.db2/.db 疑似命令において，ラベル label の ep オフセットを値として扱われます。 32 ビットの値を持ちますが，各疑似命令のビット幅に応じてマスクされます。
$label	.db4/.db2/.db 疑似命令において，ラベル label の gp オフセットを値として扱われます。 32 ビットの値を持ちますが，各疑似命令のビット幅に応じてマスクされます。

(g) ep オフセット参照

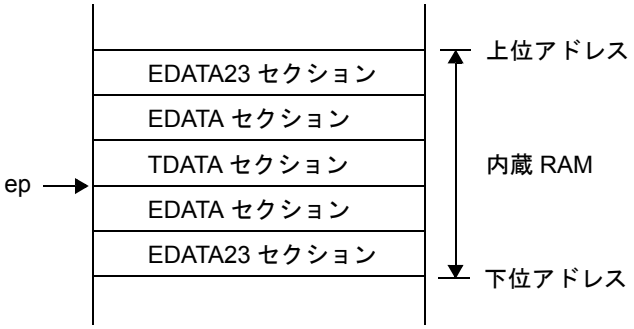
ここでは，ep オフセット参照について説明します。CC-RH では，明示的に内蔵 RAM に置かれるデータについては，基本的に，次のことが想定されています。

エレメント・ポインタ（ep）の指すアドレスからのオフセットによって参照する

なお，内蔵 RAM に置かれるデータは，次の 2 つに分けられます。

- TDATA/TDATA4/TBSS4/TDATA5/TBSS5/TDATA7/TBSS7/TDATA8/TBSS8 セクション（コード・サイズが小さいメモリ参照命令（sld/sst）で参照するデータ）
- EDATA/EBSS セクション（コード・サイズが大きいメモリ参照命令（ld/st）で参照するデータ）
- EDATA23/EBSS23 セクション（コード・サイズが大きいメモリ参照命令（ld23/st23）で参照するデータ）

図 5.2 内蔵 RAM のメモリ配置イメージ



<1> データの割り当て

内蔵 RAM に置くセクションへのデータの割り当ては，次の方法で行います。

- C 言語を用いてプログラムを作成する場合
  “#pragma section” 指令により，セクション種別 “sdata”，または “sdata23” を指定してデータを割り当てます。

- アセンブリ言語を用いてプログラムを作成する場合
  セクション定義疑似命令により，再配置属性 tdata，tdata4，tbss4，tdata5，tbss5，tdata7，tbss7，tdata8，tbss8，sdata，sbss，sdata23，または sbss23 のセクションへデータを割り当てます。

<2> データの参照

%label による参照に対しては，ep オフセット参照を行う機械語命令列が生成されます。

例

```
        .dseg   EDATA
sdata:  .db2    0xFFF0
        .dseg   DATA
data:   .db2    0xFFF0
        .cseg   TEXT
        ld.h    %sdata, r20     ; (1)
        ld.h    %data, r20      ; (2)
```

アセンブラでは，%label による参照に対して，（1），（2）の両方とも ep オフセット参照とみなし，機械語の命令列を生成します。
なお，アセンブラでは，データが配置されたセクションが正しいものとして処理が行われます。このため，データの配置に誤りがある場合でも，検出できません。

例

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

label を EDATA セクションに配置し，ep オフセット参照を行う命令を記述したが，配置誤りにより DATA セクションに配置された場合，ベース・レジスタである ep シンボル値 ＋ label の DATA セクション内オフセット値にあるデータがロードされます。

(h) gp オフセット参照

ここでは，gp オフセット参照について説明します。CC-RH では，外部 RAM に置かれるデータについては，基本的に次のことが想定されています。

グローバル・ポインタ（gp）の指すアドレスからのオフセットによって参照する

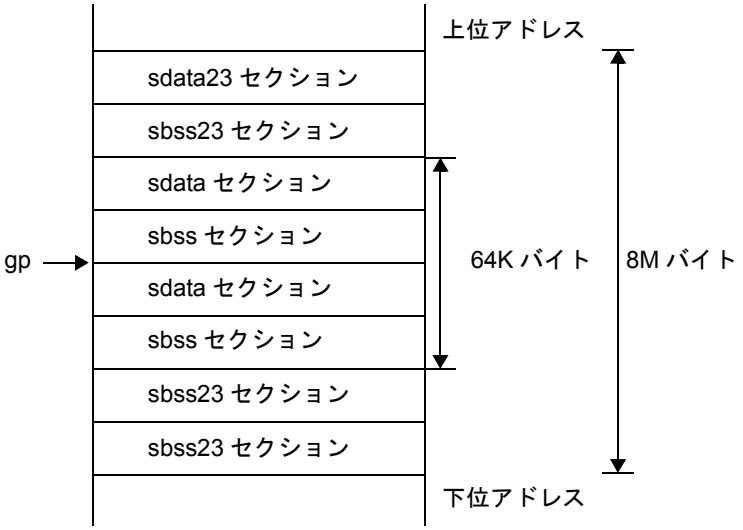
また，C 言語における “#pragma section” 指令や，アセンブリ言語におけるセクション定義疑似命令による，内蔵 ROM/ 内蔵 RAM などの r0 相対のメモリ割り当てを行わない場合，すべてのデータが gp オフセット参照となります。

<1> データの割り当て

RH850 ファミリの機械語命令におけるメモリ参照命令（ld/st）は，ディスプレースメントに 16 ビットのイミーディエトと 23 ビットのイミーディエトしかとれをとることができます。このため，CC-RH ではデータを以下に示した 3 つに分け，最初のデータが sdata セクション，または sbss セクションに，2 番目のデータが sdata23 セクション，または sbss23 セクションに，最後のデータが data セクション，または bss 属性セクションに割り当てられます。

- グローバル・ポインタ（gp）と 16 ビットのディスプレースメントを用いて参照することのできるメモリ範囲に割り当てるデータ
- グローバル・ポインタ（gp）と 23 ビットのディスプレースメントを用いて参照することのできるメモリ範囲に割り当てるデータ
- グローバル・ポインタ（gp）と（複数の命令によって構成される）32 ビットのディスプレースメントを用いて参照することのできるメモリ範囲に割り当てるデータ

図 5.3 gp オフセット参照セクションのメモリ配置イメージ



備考 sdata セクションと sbss セクションをあわせて 64K バイトです。gp は sdata セクションと sbss セクションの中心の位置です。

したがって，sdata/sbss/sdata23/sbss23 セクションのデータを参照する場合は，1 命令で行えますが，data/bss セクションのデータを参照する場合は，複数の命令が必要となります。つまり，より多くのデータを sdata/sbss/sdata23/sbss23 セクションに割り当てたほうが，生成された機械語命令の実行効率，および

オブジェクト効率は向上します。しかし，16 ビット，および 23 ビットのディスプレースメントで参照できるメモリ範囲の大きさは限られています。
そこで，すべてのデータを sdata/sbss/sdata23/sbss23 セクションに割り当てられない場合，どのデータを sdata/sbss/sdata23/sbss23 セクションに割り当てるかを定めなければなりません。
CC-RH では，“できるだけ多くのデータを sdata/sbss/sdata23/sbss23 セクションに割り当てる”という方針がとられています。デフォルトの処理では，すべてのデータが sdata/sbss/sdata23/sbss23 性セクションに割り当てられますが，割り当てられるデータを選別する場合，次の方法により行います。

- プログラムでデータを割り当てるセクションを指定する場合
  参照頻度の高いデータを，明示的に sdata/sbss/sdata23/sbss23 セクションへ割り当てます。アセンブリ言語の場合，セクション定義疑似命令で，C 言語の場合，#pragma section 指令で割り当てられます。

<2> データの参照
前述のデータの割り付けに従い，アセンブラでは，次のようになります。

- sdata/sbss 属性セクションに割り当てるデータの gp オフセット参照に対しては，16 ビットのディスプレースメントを用いた参照を行う機械語命令が生成されます。
- data/bss 属性セクションに割り当てるデータの gp オフセット参照に対しては，（複数の機械語命令によって構成される）32 ビットのディスプレースメントを用いた参照を行う機械語命令列が生成されます。

例

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

アセンブラでは，（1）で定義したデータを gp オフセット参照している（2）の ld.w 命令に対して，次の命令列に等価な機械語の命令列が生成されます。注

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

注 HIGHW1/LOWW に関しては，「(i) HIGH/LOW/HIGHW/LOWW/HIGHW1 演算子について」を参照してください。

なお，アセンブラでは，1 ファイルずつ処理が行われます。このため，指定したファイル内に定義を持つデータに対しては，そのデータがどの属性セクションに割り当てられるデータであるかを判断することができますが，指定したファイル内の定義を持たないデータに対しては判定できません。
そこで，アセンブリ言語を用いてプログラムを作成する場合，指定したファイル内に定義を持たないデータ（実際には指定したファイル内に定義を持たず gp オフセット参照されているラベル）に対し，.extern 疑似命令を用いて，サイズを指定してください。

また，C 言語を用いてプログラム作成する場合，CC-RH に含まれる C コンパイラが，指定したファイル内に定義を持たないデータ（実際には指定したファイル内に定義を持たず gp オフセット参照されているラベル）に対して，自動的に .extern 疑似命令を生成し，サイズを指定するコードを出力します。

備考 アセンブラにおけるデータの gp オフセット参照（具体的には，ラベルの gp オフセット参照を持つ相対値式をディスプレースメントとするメモリ参照命令）の扱いをまとめると，次のようになります。

- そのデータが，指定したファイル内に定義を持つ場合
  - sdata/sbss セクションに割り付けるデータである場合注
    16 ビットのディスプレースメントを用いた参照を行う機械語命令が生成されます。
  - sdata/sbss セクションに割り付けるデータでない場合
    32 ビットのディスプレースメントを用いた参照を行う機械語命令列が生成されます。

注 “ラベル ± 定数式”の形式の相対値式で，定数式の値が 16 ビットの範囲を越える場合は，アセンブラでは，32 ビットのディスプレースメントを用いた参照を行う機械語命令列が生成されます。

- そのデータが，指定したファイル内に定義を持たない場合
  sdata/sbss セクションに割り当てるデータである（gp オフセット参照されているラベルが sdata/sbss セクションに定義を持つ）とみなされ，16 ビットのディスプレースメントを用いた参照を行う機械語命令が生成されます。

(i) HIGH/LOW/HIGHW/LOWW/HIGHW1 演算子について

<1> 32 ビットの定数値をレジスタに入れる場合
アセンブラでは，32 ビットの定数値をレジスタに格納する場合，命令展開を行い movhi 命令と movea 命令を用いて，32 ビットの定数値を上位 16 ビットと下位 16 ビットに分けてレジスタに格納する命令列が生成されます。

例

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

この際，下位 16 ビットの格納に用いられる機械語の movea 命令は，指定された 16 ビットの値を符号拡張し 32 ビットの値として扱われます。この符号拡張された部分を補正するためアセンブラでは，movhi 命令を用いて上位 16 ビットをレジスタに格納する際，ただ単に上位 16 ビットをレジスタに格納するのではなく，以下の値をレジスタに格納します。

上位 16 ビット + 下位 16 ビットの最上位ビット（ビット番号 15 のビット）

<2> 32 ビットのディスプレースメントを用いてメモリを参照する場合
アセンブラでは，32 ビットのディスプレースメントを用いてメモリを参照する場合，命令展開が行われ，movhi 命令とメモリ参照命令が用いられて，32 ビットのディスプレースメントの上位 16 ビットと，下位 16 ビットから，32 ビットのディスプレースメントが構成されて参照を行う命令列が生成されます。

例

`ld.w 0x18000[r11], r12`	`movhi HIGHW1(0x18000), r11, r1` `ld.w LOWW(0x18000)[r1], r12`

この際，下位 16 ビットをディスプレースメントとして用いる機械語のメモリ参照命令は，指定された 16 ビットのディスプレースメントを符号拡張し，32 ビットの値として扱います。この符号拡張された部分を補正するために，アセンブラでは，movhi 命令を用いて上位 16 ビットのディスプレースメントを構成する際，ただ単に上位 16 ビットのディスプレースメントを構成するのではなく，次のディスプレースメントを構成します。

上位 16 ビット ＋ 下位 16 ビットの最上位ビット（ビット番号 15 のビット）

<3> HIGHW/LOWW/HIGHW1/HIGH/LOW
次表のようにアセンブラでは，HIGHW，LOWW，HIGHW1，HIGH，および LOW 演算子を用いることにより，32 ビットの値の上位 16 ビット，32 ビットの値の下位 16 ビット，および 32 ビットの値の上位 16 ビット ＋ ビット番号 15 のビット値，16 ビットの値の上位 8 ビット値，16 ビットの値の下位 8 ビット値を指定できます。[注]

注 アセンブラ内部では解決できない場合，この情報はリロケーション情報に反映されリンク・エディタにおいて解決されます。

表 5.26 領域確保疑似命令

HIGHW/LOWW/HIGHW1/HIGH/LOW	意味
HIGHW (*value*)	*value* の上位 16 ビット
LOWW (*value*)	*value* の下位 16 ビット
HIGHW1 (*value*)	*value* の上位 16 ビット ＋ *value* のビット番号 15 のビット値
HIGH (*value*)	*value* の下位 16 ビット中の上位 8 ビット
LOW (*value*)	*value* の下位 8 ビット

例

```
        .dseg   DATA
L1:
            :
        .cseg   TEXT
        movhi   HIGHW ($L1), r0, r10    ;L1 の gp オフセットの値の上位 16 ビットを
                                        ;r10 の上位 16 ビットに格納し，
                                        ; 位 16 ビットに 0 を格納する。
        movea   LOWW ($L1), r0, r10     ;L1 の gp オフセットの値の下位 16 ビットを
                                        ; 符号拡張し r10 に格納する。
            :
        movhi   HIGHW1 ($L1), r0, r1    ;L1 の gp オフセットの値を r10 に格納する。
        movea   LOWW ($L1), r1, r10
```

5.9 命令の説明

アセンブラがサポートするアセンブラ命令について，次の形式で説明します。

なお，アセンブラが生成する機械語命令についての詳細は，各デバイスのユーザーズ・マニュアルを参照してください。

命令

命令の意味を日本語と英語で示します。

[指定形式]

命令の形式を示します。

[機能]

命令の機能を示します。

[詳細説明]

命令の動作の仕方を示します。

[フラグ]

命令実行によるフラグ（PSW）の動きを示します。
ただし，ビット操作命令（set1，clr1，not1）では，実行前のフラグの値を示しています。
なお，表中の “—” は，フラグの値が変化しないことを示します。

[注意事項]

命令における注意事項を示します。

5.9.1 ロード／ストア命令

この項では，ロード／ストア命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.27 ロード／ストア命令

命令		意味
ld	ld.b	バイト・データのロード
	ld.h	ハーフワード・データのロード
	ld.w	ワード・データのロード
	ld.bu	符号なしバイト・データのロード
	ld.hu	符号なしハーフワード・データのロード
sld	sld.b	バイト・データのロード（ショート・フォーマット）
	sld.h	ハーフワード・データのロード（ショート・フォーマット）
	sld.w	ワード・データのロード（ショート・フォーマット）
	sld.bu	符号なしバイト・データのロード（ショート・フォーマット）
	sld.hu	符号なしハーフワード・データのロード（ショート・フォーマット）
ld23	ld23.b	バイト・データのロード
	ld23.h	ハーフワード・データのロード
	ld23.w	ワード・データのロード
	ld23.bu	符号なしバイト・データのロード
	ld23.hu	符号なしハーフワード・データのロード
	ld23.dw	ダブルワード・データのロード
st	st.b	バイト・データのストア
	st.h	ハーフワード・データのストア
	st.w	ワード・データのストア
sst	sst.b	バイト・データのストア（ショート・フォーマット）
	sst.h	ハーフワード・データのストア（ショート・フォーマット）
	sst.w	ワード・データのストア（ショート・フォーマット）
st23	st23.b	バイト・データのストア
	st23.h	ハーフワード・データのストア
	st23.w	ワード・データのストア
	st23.dw	ダブルワード・データのストア

ld

データのロードを行います。（Load）

［指定形式］

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

disp に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに LOWW を適用したもの

［機能］

ld.b，ld.h，ld.w，ld.bu，ld.hu 命令は，第 1 オペランドに指定したアドレスから 1 バイト分，1 ハーフワード分，および 1 ワード分のデータを取り込み，第 2 オペランドに指定したレジスタにロードします。

［詳細説明］

- disp に次のものを指定した場合，アセンブラでは，機械語命令の ld 命令[注]が 1 つ生成されます。なお，例中の “ld” は ld.b，ld.h，ld.w，ld.bu，ld.hu のいずれかになります。

(a) -32768 ～ +32767 の範囲の絶対値式

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

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(c) !label，または %label を持つ絶対値式

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

(d) LOWW を適用したもの

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

注 機械語命令の ld 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp を省略した場合，アセンブラでは，0 を指定したものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に LOWW を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に LOWW を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合アセンブラでは，[gp] が指定されたものとみなされます。
- 23 ビット幅の disp をもつ ld 命令を指定したい場合は，ld23 命令を指定してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- ld.b，および ld.h は，1 バイト分，および 1 ハーフワード分のデータを符号拡張し，1 ワード分のデータとしてレジスタにロードされます。
- ld.h，ld.w，ld.hu の disp に 2 の倍数でない値を指定した場合，アセンブラ，またはリンク・エディタでは，disp に対して，2 でアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

- ld.bu，および ld.hu 命令に対し，第 2 オペランドに r0 を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

sld

データのロードを行います。（Short format Load）

[指定形式]

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

disp4/5/7/8 に指定できるものを次に示します。

- sld.b では 7 ビット，sld.h，および sld.w では 8 ビット，sld.bu では 4 ビット，sld.hu では 5 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

sld.b，sld.h，sld.w，sld.bu，sld.hu 命令は，第 1 オペランドに指定したディスプレースメントとレジスタ ep の内容を加算して得たアドレスから，1 バイト分，1 ハーフワード分，および 1 ワード分のデータを取り込み，第 2 オペランドに指定したレジスタにロードします。

[詳細説明]

アセンブラでは，機械語命令の sld 命令が 1 つ生成されます。
ベース・レジスタの指定 “[ep]” は省略できます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- sld.b，および sld.h は，1 バイト分，および 1 ハーフワード分のデータを符号拡張し，1 ワードのデータとしてレジスタに格納されます。
- sld.h/sld.hu の disp8/disp5 に 2 の倍数でない値を指定した場合，および sld.w の disp8 に 4 の倍数でない値を指定した場合，アセンブラでは，disp8/disp5 に対して，それぞれ 2 の倍数，4 の倍数にアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

- sld.b の disp7 に 127 を越える値を指定した場合，sld.h，sld.w の disp8 に 255 を越える値を指定した場合，sld.bu の disp4 に 16 を越える値を指定した場合，および sld.hu の disp5 に 32 を越える値を指定した場合，次のメッセージが出力され，disp7，disp8，disp4，disp5 をそれぞれ 0x7F，0xFF，0xF，0x1F でマスクしたコードが生成されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

- sld.bu，および sld.hu の第 2 オペランド（reg2）に r0 を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

ld23

データのロードを行います。（Load）

［指定形式］

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

disp23 に指定可能なものを以下に示します。

- 23 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

ld23.b，ld23.h，ld23.w，ld23.bu，ld23.hu 命令は，第 1 オペランドに指定したアドレスから 1 バイト分，1 ハーフワード分，および 1 ワード分のデータを取り込み，第 2 オペランドに指定したレジスタにロードします。ld23.dw 命令は，第 1 オペランドに指定したアドレスからダブルワード・データ分のデータを取り込み，第 2 オペランドに指定したレジスタ reg2 に下位 32 ビット，reg2 ＋ 1 に上位 32 ビットをロードします。

［詳細説明］

- アセンブラでは，機械語命令の 48 ビット長 ld 命令[注]が 1 つ生成されます。

注　機械語命令の ld23 命令は，ディスプレースメントに -4194304 ～ +4194303（0xFFC00000 ～ 0x3FFFFF）の範囲のイミーディエトをとります。

- disp23 を省略した場合，アセンブラでは，0 を指定したものとみなされます。
- disp23 に #label を持つ相対値式を指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp23 に $label を持つ相対値式を指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- ld23.b，および ld23.h は，1 バイト分，および 1 ハーフワード分のデータを符号拡張し，1 ワード分のデータとしてレジスタにロードされます。
- ld23.h，ld23.w，ld23.hu，ld23.dw の disp23 に 2 の倍数でない値を指定した場合，アセンブラ，またはリンク・エディタでは，disp23 に対して，2 でアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

- ld23.bu，および ld23.hu 命令に対し，第 2 オペランドに r0 を指定した場合，メッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

- ld23.dw 命令に対し，第 2 オペランドに奇数番号のレジスタを指定した場合，次のメッセージが出力されます。

W0550013 : *register* が、*kind* レジスタとしてオペランドに指定されています。

st

データのストアを行います。（Store）

［指定形式］

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

disp に指定できるものを，次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに LOWW を適用したもの

［機能］

st.b，st.h，st.w 命令は，第 1 オペランドに指定したレジスタの下位 1 バイト分，下位 1 ハーフワード分，および 1 ワード分のデータを取り込み，第 2 オペランドに指定したアドレスにストアします。

［詳細説明］

- disp に次のものを指定した場合，アセンブラでは，機械語命令の st 命令[注]が 1 つ生成されます。なお，例中の “st” は st.b，st.h，st.w のいずれかになります。

(a) -32768 ～ +32767 の範囲の絶対値式

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

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(c) !label，または %label を持つ絶対値式

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

(d) LOWW を適用したもの

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

注 機械語命令の st 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp を省略した場合，アセンブラでは，0 を指定したものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に LOWW を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に LOWW を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。
- 23 ビット幅の disp をもつ st 命令を指定したい場合は，st23 命令を指定してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- st.h，および st.w の disp に 2 の倍数でない値を指定した場合，アセンブラでは，disp に対して 2 でアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

sst

データのストアを行います。（Short format Store）

[指定形式]

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

disp7/8 に指定できるものを次に示します。

- sst.b では 7 ビット，sst.h，および sst.w では 8 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

sst.b，sst.h，sst.w 命令は，第 1 オペランドに指定したレジスタの下位 1 バイト分，下位 1 ハーフワード分，および 1 ワード分のデータを，第 2 オペランドに指定したディスプレースメントとしてレジスタ ep の内容を加算して得たアドレスに格納します。

[詳細説明]

アセンブラでは，機械語命令の sst 命令が 1 つ生成されます。
ベース・レジスタの指定 “[ep]” は省略できます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- sst.h の disp8 に 2 の倍数でない値を指定した場合，および sst.w の disp8 に 4 の倍数でない値を指定した場合，アセンブラでは，disp8 に対して，それぞれ 2 の倍数，4 の倍数にアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

- sst.b の disp7 に 127 を越える値を指定した場合，および sst.h，sst.w の disp8 に 255 を越える値を指定した場合，次のメッセージが出力され，disp7，disp8 をそれぞれ 0x7F，0xFF でマスクしたコードが生成されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

st23

データのストアを行います。（Store）

［指定形式］

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

disp23 に指定できるものを，次に示します。

- 23 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

st23.b，st23.h，st23.w 命令は，第 1 オペランドに指定したレジスタの下位 1 バイト分，下位 1 ハーフワード分，および 1 ワード分のデータを取り込み，第 2 オペランドに指定したアドレスにストアします。st23.dw 命令は，第 1 オペランドに指定したレジスタのワード・データを下位 32 ビット，reg2 ＋ 1 のワード・データを上位 32 ビットとするダブルワード・データを，第 2 オペランドに指定したアドレスにストアします。

［詳細説明］

- アセンブラでは，機械語命令の st 命令[注]が 1 つ生成されます。

注　機械語命令の st 命令は，ディスプレースメントに -4194304 ～ +4194303（0xFFC00000 ～ 0x3FFFFF）の範囲のイミーディエトをとります。

- disp23 を省略した場合，アセンブラでは，0 を指定したものとみなされます。
- disp23 に #label を持つ相対値式を指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp23 に $label を持つ相対値式を指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- st23.h，および st23.w の disp23 に 2 の倍数でない値を指定した場合，アセンブラでは，disp23 に対して 2 でアライメントしたコードが生成され，次のいずれかのメッセージが出力されます。

W0550010 : ディスプレースメントの値が指定可能な値の範囲を越えています。
E0562332 : Relocation value is odd number : " *ファイル* "-" *セクション* "-" *オフセット* "

- st23.dw 命令に対し，第 1 オペランドに奇数番号のレジスタを指定した場合，メッセージが出力されます。

W0550013 : *register* が、*kind* レジスタとしてオペランドに指定されています。

5.9.2 算術演算命令

この項では，算術演算命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.28 算術演算命令

命令	意味
add	加算
addi	加算（イミーディエト）
adf	条件付き加算
sub	減算
subr	逆減算
sbf	条件付き減算
mulh	符号付き乗算（ハーフワード）
mulhi	符号付き乗算（ハーフワード・イミーディエト）
mul	符号付き乗算（ワード）
mulu	符号なし乗算
mac	符号付きワード・データの加算付き乗算
macu	符号なしワード・データの加算付き乗算
divh	符号付き除算（ハーフワード）
div	符号付き除算（ワード）
divhu	符号なし除算（ハーフワード）
divu	符号なし除算（ワード）
divq	符号付きワード・データの除算（可変ステップ）
divqu	符号なしワード・データの除算（可変ステップ）
cmp	比較
mov	データの転送
movea	実行アドレスの転送
movhi	上位ハーフワードの転送
mov32	32 ビット・データの転送
cmov	フラグ条件付きデータの転送
setf	フラグ条件の設定
sasf	論理左シフト付きフラグ条件の設定
bins	レジスタ上のビット挿入
rotl	ローテート

add

加算を行います。（Add）

［指定形式］

- add reg1, reg2
- add imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

- “add reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値を加算し，結果を第 2 オペランドに指定したレジスタに格納します。
- “add imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と第 2 オペランドに指定したレジスタ値を加算し，結果を第 2 オペランドに指定したレジスタに格納します。

［詳細説明］

- “add reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の add 命令が 1 つ生成されます。
- “add imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の add 命令[注]が 1 つ生成されます。

(a) -16 ～ +15 の範囲の絶対値式

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

注 機械語命令の add 命令は，第 1 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “add imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`add imm16, reg`	`addi imm16, reg, reg`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`add imm, reg`	`mov imm, r1` `add r1, reg`

(c) !label，または %label を持つ相対値式，および sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`add !label, reg`	`addi !label, reg, reg`
`add %label, reg`	`addi %label, reg, reg`
`add $label, reg`	`addi $label, reg, reg`

(d) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

［フラグ］

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

addi

加算を行います。（Add Immediate）

［指定形式］

- addi imm, reg1, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

［機能］

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値と第 2 オペランドに指定したレジスタ値を加算し，結果を第 3 オペランドに指定したレジスタに格納します。

［詳細説明］

- imm に次のものを指定した場合，アセンブラでは，機械語命令の addi 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

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

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(c) !label，または %label を持つ相対値式

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

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

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

注 機械語命令の addi 命令は，第 1 オペランドに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

<1> imm の値の下位 16 ビットがすべて 0 の場合
reg2 が r0 の場合

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

上記以外の場合

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

<2> 上記以外の場合
reg2 が r0 の場合

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

上記以外の場合

addi imm, reg1, reg2	mov imm, reg2 add reg1, reg2

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式
reg2 が r0 の場合

addi #label, reg1, r0	mov #label, r1 add reg1, r1
addi label, reg1, r0	mov label, r1 add reg1, r1
addi $label, reg1, r0	mov $label, r1 add reg1, r1

上記以外の場合

addi #label, reg1, reg2	mov #label, reg2 add reg1, reg2
addi label, reg1, reg2	mov label, reg2 add reg1, reg2
addi $label, reg1, reg2	mov $label, reg2 add reg1, reg2

［フラグ］

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

adf

条件付き加算を行います。（Add on Condition Flag）

[指定形式]

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

imm4 に指定できるものを次に示します。

- 4 ビット幅までの値を持つ絶対値式（0xD は指定できません）

[機能]

- “adf imm4, reg1, reg2, reg3” の形式
  第 1 オペランドに指定した絶対値式の下位 4 ビットの値で示されるフラグ状態（「表 5.29 adfcnd 命令」を参照）と現在のフラグ状態を比較します。
  値が一致した場合には，第 3 オペランドに指定したレジスタのワード・データに，第 2 オペランドに指定したレジスタのワード・データを加算し，さらに 1 を加算します。結果を第 4 オペランドに指定したレジスタに格納します。
  値が一致しなかった場合には，第 3 オペランドに指定したレジスタのワード・データに，第 2 オペランドに指定したレジスタのワード・データを加算します。その結果を第 4 オペランドに指定したレジスタに格納します。
- “adf*cnd* reg1, reg2, reg3” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較します。
  値が一致した場合に第 2 オペランドに指定したレジスタのワード・データに，第 1 オペランドに指定したレジスタのワード・データを加算し，さらに 1 を加算します。その結果を第 3 オペランドに指定したレジスタに格納します。
  値が一致しなかった場合には，第 2 オペランドに指定したレジスタのワード・データに，第 1 オペランドに指定したレジスタのワード・データを加算します。その結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

- adf 命令に対し，アセンブラでは，機械語命令の adf 命令が 1 つ生成されます。
- adf*cnd* 命令に対し，アセンブラでは，対応する adf 命令が生成され（「表 5.29 adfcnd 命令」を参照），“adf imm4, reg1, reg2, reg3” の形式に展開されます。

表 5.29 adf*cnd* 命令

命令	フラグ状態	フラグ状態の意味	命令展開
adfgt	((S xor OV) or Z) = 0	Greater than（signed）	adf 0xF
adfge	(S xor OV) = 0	Greater than or equal（signed）	adf 0xE
adflt	(S xor OV) = 1	Less than（signed）	adf 0x6
adfle	((S xor OV) or Z) = 1	Less than or equal（signed）	adf 0x7
adfh	(CY or Z) = 0	Higher（Greater than）	adf 0xB
adfnl	CY = 0	Not lower（Greater than or equal）	adf 0x9
adfl	CY = 1	Lower（Less than）	adf 0x1
adfnh	(CY or Z) = 1	Not higher（Less than or equal）	adf 0x3
adfe	Z = 1	Equal	adf 0x2
adfne	Z = 0	Not equal	adf 0xA

命令	フラグ状態	フラグ状態の意味	命令展開
adfv	OV = 1	Overflow	adf 0x0
adfnv	OV = 0	No overflow	adf 0x8
adfn	S = 1	Negative	adf 0x4
adfp	S = 0	Positive	adf 0xC
adfc	CY = 1	Carry	adf 0x1
adfnc	CY = 0	No carry	adf 0x9
adfz	Z = 1	Zero	adf 0x2
adfnz	Z = 0	Not zero	adf 0xA
adft	always 1	Always 1	adf 0x5

[フラグ]

CY	MSB（Most Significant Bit）からのキャリーがあれば 1，そうでない場合 0
OV	オーバーフローが起こった場合 1，そうでない場合 0
S	演算結果が負の場合 1，そうでない場合 0
Z	演算結果が 0 の場合 1，そうでない場合 0
SAT	—

[注意事項]

- adf 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定された値の下位 4 ビットが用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

- adf 命令の imm4 に 0xD を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550261 : 指定された条件コードが不正です。

sub

減算を行います。(Subtract)

[指定形式]

- sub reg1, reg2
- sub imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “sub reg1, reg2” の形式
  第 2 オペランドに指定したレジスタ値から，第 1 オペランドに指定したレジスタ値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。
- “sub imm, reg2” の形式
  第 2 オペランドに指定したレジスタ値から，第 1 オペランドに指定した絶対値式，または相対値式の値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “sub reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の sub 命令が 1 つ生成されます。
- “sub imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

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

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

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

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

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

(d) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

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

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の sub 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

subr

逆減算を行います。(Subtract Reverse)

[指定形式]

- subr reg1, reg2
- subr imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “subr reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値から，第 2 オペランドに指定したレジスタ値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。
- “subr imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値から，第 2 オペランドに指定したレジスタ値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “subr reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の subr 命令が 1 つ生成されます。
- “subr imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

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

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

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

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

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

(d) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

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

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の subr 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

sbf

条件付き減算を行います。（Subtract on Condition Flag）

［指定形式］

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

imm4 に指定できるものを次に示します。

- 4 ビット幅までの値を持つ絶対値式（0xD は指定できません）

［機能］

- “sbf imm4, reg1, reg2, reg3” の形式
  第 1 オペランドに指定した絶対値式の下位 4 ビットの値で示されるフラグ状態（「表 5.30 sbfcnd 命令」を参照）と現在のフラグ状態を比較します。
  値が一致した場合には，第 3 オペランドに指定したレジスタのワード・データから，第 2 オペランドに指定したレジスタのワード・データを減算し，さらに 1 を減算します。その結果を第 4 オペランドに指定したレジスタに格納します。
  値が一致しなかった場合には，第 3 オペランドに指定したレジスタのワード・データから，第 2 オペランドに指定したレジスタのワード・データを減算します。その結果を第 4 オペランドに指定したレジスタに格納します。
- “sbf*cnd* reg1, reg2, reg3” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較します。
  値が一致した場合には，第 2 オペランドに指定したレジスタのワード・データから，第 1 オペランドに指定したレジスタのワード・データを減算し，さらに 1 を減算します。その結果を第 3 オペランドに指定したレジスタに格納します。
  値が一致しなかった場合には，第 2 オペランドに指定したレジスタのワード・データから，第 1 オペランドに指定したレジスタのワード・データを減算します。その結果を第 3 オペランドに指定したレジスタに格納します。

［詳細説明］

- sbf 命令に対し，アセンブラでは，機械語命令の sbf 命令が 1 つ生成されます。
- sbf*cnd* 命令に対し，アセンブラでは，対応する sbf 命令が生成され（「表 5.30 sbfcnd 命令」を参照），“sbf imm4, reg1, reg2, reg3” の形式に展開されます。

表 5.30 sbf*cnd* 命令

命令	フラグ状態	フラグ状態の意味	命令展開
sbfgt	((S xor OV) or Z) = 0	Greater than（signed）	sbf 0xF
sbfge	(S xor OV) = 0	Greater than or equal（signed）	sbf 0xE
sbflt	(S xor OV) = 1	Less than（signed）	sbf 0x6
sbfle	((S xor OV) or Z) = 1	Less than or equal（signed）	sbf 0x7
sbfh	(CY or Z) = 0	Higher（Greater than）	sbf 0xB
sbfnl	CY = 0	Not lower（Greater than or equal）	sbf 0x9
sbfl	CY = 1	Lower（Less than）	sbf 0x1
sbfnh	(CY or Z) = 1	Not higher（Less than or equal）	sbf 0x3
sbfe	Z = 1	Equal	sbf 0x2
sbfne	Z = 0	Not equal	sbf 0xA

命令	フラグ状態	フラグ状態の意味	命令展開
sbfv	OV = 1	Overflow	sbf 0x0
sbfnv	OV = 0	No overflow	sbf 0x8
sbfn	S = 1	Negative	sbf 0x4
sbfp	S = 0	Positive	sbf 0xC
sbfc	CY = 1	Carry	sbf 0x1
sbfnc	CY = 0	No carry	sbf 0x9
sbfz	Z = 1	Zero	sbf 0x2
sbfnz	Z = 0	Not zero	sbf 0xA
sbft	always 1	Always 1	sbf 0x5

[フラグ]

CY	MSB（Most Significant Bit）からのボローがあれば 1，そうでない場合 0
OV	オーバーフローが起こった場合 1，そうでない場合 0
S	演算結果が負の場合 1，そうでない場合 0
Z	演算結果が 0 の場合 1，そうでない場合 0
SAT	—

[注意事項]

- sbf 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定された値の下位 4 ビットが用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

- sbf 命令の imm4 に 0xD を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550261 : 指定された条件コードが不正です。

mulh

符号付き乗算（ハーフワード）を行います。（Multiply Half-word）

[指定形式]

- mulh reg1, reg2
- mulh imm, reg2

imm に指定可能なものを以下に示します。

- 16 ビット幅までの値を持つ絶対値式[注]
- 相対値式

注 アセンブラでは，16 ビットを越えるかどうかチェックは行われません。生成された機械語命令 mulh 命令では，下位 16 ビットを用いて演算が行われます。

[機能]

- “mulh reg1, reg2” の形式
  第 1 オペランドに指定したレジスタの下位ハーフワード・データの値と，第 2 オペランドに指定したレジスタの下位ハーフワード・データの値を，符号付きの値として乗算し，結果を第 2 オペランドに指定したレジスタに格納します。
- “mulh imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の下位ハーフワード・データの値と，第 2 オペランドに指定したレジスタの下位ハーフワード・データの値を，符号付きの値として乗算し，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “mulh reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の mulh 命令が 1 つ生成されます。
- “mulh imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の mulh 命令[注]が 1 つ生成されます。

  (a) -16 ～ +15 の範囲の絶対値式

`mulh imm5, reg`	`mulh imm5, reg`

注 機械語命令の mulh 命令は，第 1 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “mulh imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

  (a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`mulh imm16, reg`	`mulhi imm16, reg, reg`

  (b) -32768 ～ +32767 の範囲を越える絶対値式
  imm の値の下位 16 ビットがすべて 0 の場合

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

  上記以外の場合

`mulh imm, reg`	`mov imm, r1` `mulh r1, reg`

(c) !label，または %label を持つ相対値式，および sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`mulh !label, reg`	`mulhi !label, reg, reg`
`mulh %label, reg`	`mulhi %label, reg, reg`
`mulh $label, reg`	`mulhi $label, reg, reg`

(d) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- 第 2 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

mulhi

符号付き乗算（ハーフワード）を行います。(Multiply Half-word)

[指定形式]

- mulhi imm, reg1, reg2

imm に指定可能なものを以下に示します。

- 16 ビット幅までの値を持つ絶対値式[注]
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

注 アセンブラでは，16 ビットを越えるかどうかチェックは行われません。生成された機械語命令 mulhi 命令では，下位 16 ビットを用いて演算が行われます。

[機能]

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値と，第 2 オペランドに指定したレジスタ値を符号付きの値として乗算し，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

- imm に次のものを指定した場合，アセンブラでは，機械語命令の mulhi 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`mulhi imm16, reg1, reg2`	`mulhi imm16, reg1, reg2`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`mulhi $label, reg1, reg2`	`mulhi $label, reg1, reg2`

(c) !label，または %label を持つ相対値式

`mulhi !label, reg1, reg2`	`mulhi !label, reg1, reg2`
`mulhi %label, reg1, reg2`	`mulhi %label, reg1, reg2`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`mulhi imm16, reg1, reg2`	`mulhi imm16, reg1, reg2`

注 機械語命令の mulhi 命令は，第 1 オペランドに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`mulhi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `mulh reg1, reg2`

上記以外の場合

`mulhi imm, reg1, reg2`	`mov imm, reg2` `mulh reg1, reg2`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`mulhi #label, reg1, reg2`	`mov #label, reg2` `mulh reg1, reg2`
`mulhi label, reg1, reg2`	`mov label, reg2` `mulh reg1, reg2`
`mulhi $label, reg1, reg2`	`mov $label, reg2` `mulh reg1, reg2`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- 第 3 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

mul

符号付き乗算（ワード）を行います。（Multiply Word）

[指定形式]

- mul reg1, reg2, reg3
- mul imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “mul reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値を，符号付きの値として乗算し，結果の下位 32 ビットを第 2 オペランドに指定したレジスタに，上位 32 ビットを第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには乗算結果の上位 32 ビットを格納します。
- “mul imm, reg2, reg3” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタの値を，符号付きの値として乗算し，結果の下位 32 ビットを第 2 オペランドに指定したレジスタに，上位 32 ビットを第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには乗算結果の上位 32 ビットを格納します。

[詳細説明]

- “mul reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の mul 命令が 1 つ生成されます。
- “mul imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) 0

`mul 0, reg2, reg3`	`mul r0, reg2, reg3`

(b) 0 以外の -256 ～ +255 の範囲の絶対値式

`mul imm9, reg2, reg3`	`mul imm9, reg2, reg3`

(c) -256 ～ +255 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`mul imm16, reg2, reg3`	`movea imm16, r0, r1` `mul r1, reg2, reg3`

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`mul imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mul r1, reg2, reg3`

上記以外の場合

`mul imm, reg2, reg3`	`mov imm, r1` `mul r1, reg2, reg3`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`mul  $label, reg2, reg3`	`movea  $label, r0, r1` `mul  r1, reg2, reg3`

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`mul  #label, reg2, reg3`	`mov  #label, r1` `mul  r1, reg2, reg3`
`mul  label, reg2, reg3`	`mov  label, r1` `mul  r1, reg2, reg3`
`mul  $label, reg2, reg3`	`mov  $label, r1` `mul  r1, reg2, reg3`

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

mulu

符号なし乗算（ワード）を行います。（Multiply Word Unsigned）

[指定形式]

- mulu reg1, reg2, reg3
- mulu imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “mulu reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値を，符号なしの値として乗算し，結果の下位 32 ビットを第 2 オペランドに指定したレジスタに，上位 32 ビットを第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには結果の上位 32 ビットを格納します。
- “mulu imm, reg2, reg3” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタの値を，符号なしの値として乗算し，結果の下位 32 ビットを第 2 オペランドに指定したレジスタに，上位 32 ビットを第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには結果の上位 32 ビットを格納します。

[詳細説明]

- “mulu reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の mulu 命令が 1 つ生成されます。
- “mulu imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) 0

`mulu 0, reg2, reg3`	`mulu r0, reg2, reg3`

(b) 0 以外の 0 ～ 511 の範囲

`mulu imm9, reg2, reg3`	`mulu imm9, reg2, reg3`

(c) 0 ～ 511 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`mulu imm16, reg2, reg3`	`movea imm16, r0, r1` `mulu r1, reg2, reg3`

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`mulu imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `mulu r1, reg2, reg3`

上記以外の場合

`mulu imm, reg2, reg3`	`mov imm, r1` `mulu r1, reg2, reg3`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`mulu  $label, reg2, reg3`	`movea  $label, r0, r1` `mulu  r1, reg2, reg3`

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`mulu  #label, reg2, reg3`	`mov  #label, r1` `mulu  r1, reg2, reg3`
`mulu  label, reg2, reg3`	`mov  label, r1` `mulu  r1, reg2, reg3`
`mulu  $label, reg2, reg3`	`mov  $label, r1` `mulu  r1, reg2, reg3`

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

mac

符号付きワード・データの加算付き乗算を行います。（Multiply Word and Add）

[指定形式]

- mac reg1, reg2, reg3, reg4

[機能]

汎用レジスタ reg2 のワード・データに，汎用レジスタ reg1 のワード・データを乗算した結果（64 ビット・データ）と，汎用レジスタ reg3 を下位 32 ビットとして，汎用レジスタ reg3+1（たとえば，reg3 が r6 の場合，「reg3+1」は r7 となります）を上位 32 ビットとして結合した 64 ビット・データを加算し，その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg4+1 に，下位 32 ビットを汎用レジスタ reg4 に格納します。
汎用レジスタ reg1，reg2 の内容を 32 ビットの符号付き整数として扱います。
汎用レジスタ reg1，reg2，reg3，reg3+1 は影響を受けません。

[詳細説明]

アセンブラでは，機械語命令の mac 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- reg3，または reg4 に指定できる汎用レジスタは，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）だけです。
  奇数番号の付いたレジスタ（r1，r3，…，r31）を指定した場合は，次のメッセージが出力され，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）を指定したとして，アセンブルが続行されます。

  W0550026 : 奇数番号の付いたレジスタ（r*XX*）が指定されています。
  偶数番号の付いたレジスタ（r*YY*）を指定したとして、アセンブルを続行します。

macu

符号なしワード・データの加算付き乗算を行います。(Multiply Word Unsigned and Add)

[指定形式]

- macu reg1, reg2, reg3, reg4

[機能]

汎用レジスタ reg2 のワード・データに，汎用レジスタ reg1 のワード・データを乗算した結果（64 ビット・データ）と，汎用レジスタ reg3 を下位 32 ビットとして，汎用レジスタ reg3+1（たとえば，reg3 が r6 の場合，「reg3+1」は r7 となります）を上位 32 ビットとして結合した 64 ビット・データを加算し，その結果（64 ビット・データ）の上位 32 ビットを汎用レジスタ reg4+1 に，下位 32 ビットを汎用レジスタ reg4 に格納します。
汎用レジスタ reg1，reg2 の内容を 32 ビットの符号なし整数として扱います。
汎用レジスタ reg1，reg2，reg3，reg3+1 は影響を受けません。

[詳細説明]

アセンブラでは，機械語命令の macu 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- reg3，または reg4 に指定できる汎用レジスタは，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）だけです。
  奇数番号の付いたレジスタ（r1，r3，…，r31）を指定した場合は，次のメッセージが出力され，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）を指定したとして，アセンブルが続行されます。

```
W0550026 : 奇数番号の付いたレジスタ（rXX）が指定されています。
偶数番号の付いたレジスタ（rYY）を指定したとして、アセンブルを続行します。
```

divh

符号付き除算（ハーフワード）を行います。（Divide Half-word）

[指定形式]

- divh reg1, reg2
- divh imm, reg2
- divh reg1, reg2, reg3
- divh imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 16 ビット幅までの値を持つ絶対値式[注]
- 相対値式

注 アセンブラでは，16 ビットを越えるかどうかのチェックは行われません。生成された機械語命令では，下位 16 ビットを用いて演算が行われます。

[機能]

- “divh reg1, reg2” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定したレジスタの下位ハーフワード・データの値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに格納します。
- “divh imm, reg2” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定した絶対値式，または相対値式の下位ハーフワード・データの値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに格納します。
- “divh reg1, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定したレジスタの下位ハーフワード・データの値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。
- “divh imm, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定した絶対値式，または相対値式の下位ハーフワード・データの値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。

[詳細説明]

- “divh reg1, reg2”，および “divh reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の divh 命令が 1 つ生成されます。
- “divh imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。[注]

(a) 0 以外の -16 ～ +15 の範囲の絶対値式

`divh imm5, reg`	`mov imm5, r1` `divh r1, reg`

(b) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`divh imm16, reg`	`movea imm16, r0, r1` `divh r1, reg`

(c) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

divh imm, reg	movhi HIGHW(imm), r0, r1 divh r1, reg

上記以外の場合

divh imm, reg	mov imm, r1 divh r1, reg

(d) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

divh $label, reg2	movea $label, r0, r1 divh r1, reg

(e) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

divh #label, reg	mov #label, r1 divh r1, reg
divh label, reg	mov label, r1 divh r1, reg
divh $label, reg	mov $label, r1 divh r1, reg

注 機械語命令の divh 命令は，オペランドにイミーディエトをとりません。

- “divh imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) 0

divh 0, reg2, reg3	divh r0, reg2, reg3

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

divh imm5, reg2, reg3	mov imm5, r1 divh r1, reg2, reg3

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

divh imm16, reg2, reg3	movea imm16, r0, r1 divh r1, reg2, reg3

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

divh imm, reg2, reg3	movhi HIGHW(imm), r0, r1 divh r1, reg2, reg3

上記以外の場合

divh imm, reg2, reg3	mov imm, r1 divh r1, reg2, reg3

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

divh $label, reg2, reg3	movea $label, r0, r1 divh r1, reg2, reg3

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義に持たないラベルの $label を持つ相対値式

`divh #label, reg2, reg3`	`mov #label, r1` `divh r1, reg2, reg3`
`divh label, reg2, reg3`	`mov label, r1` `divh r1, reg2, reg3`
`divh $label, reg2, reg3`	`mov $label, r1` `divh r1, reg2, reg3`

[フラグ]

CY	—
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- “divh reg1, reg2” の第 1 オペランド（reg1）に r0 を指定すると，次のいずれかメッセージが出力され，アセンブルが中止されます。

E0550239 : RH850 指定時には、ソース・オペランドに r0 を指定することはできません。
E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

- “divh imm, reg2” の第 2 オペランド（reg2）に r0 を指定すると，次のいずれかメッセージが出力され，アセンブルが中止されます。

E0550239 : RH850 指定時には、ソース・オペランドに r0 を指定することはできません。
E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

- “divh imm, reg2” の第 2 オペランド（imm）に 0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550239 : RH850 指定時には、ソース・オペランドに r0 を指定することはできません。

div

符号付き除算（ワード）を行います。（Divide Word）

[指定形式]

- div reg1, reg2, reg3
- div imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “div reg1, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定したレジスタ値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに，それぞれ格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。
- “div imm, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定した絶対値式，または相対値式の値で符号付きの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。

[詳細説明]

- “div reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の div 命令が 1 つ生成されます。
- “div imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。[注]

(a) 0

`div 0, reg2, reg3`	`div r0, reg2, reg3`

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

`div imm5, reg2, reg3`	`mov imm5, r1` `div r1, reg2, reg3`

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`div imm16, reg2, reg3`	`movea imm16, r0, r1` `div r1, reg2, reg3`

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`div imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `div r1, reg2, reg3`

上記以外の場合

`div imm, reg2, reg3`	`mov imm, r1` `div r1, reg2, reg3`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`div  $label, reg2, reg3`	`movea  $label, r0, r1` `div  r1, reg2, reg3`

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義に持たないラベルの $label を持つ相対値式

`div  #label, reg2, reg3`	`mov  #label, r1` `div  r1, reg2, reg3`
`div  label, reg2, reg3`	`mov  label, r1` `div  r1, reg2, reg3`
`div  $label, reg2, reg3`	`mov  $label, r1` `div  r1, reg2, reg3`

注 機械語命令の div 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

divhu

符号なし除算（ハーフワード）を行います。（Divide Half-word Unsigned）

[指定形式]

- divhu reg1, reg2, reg3
- divhu imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 16 ビット幅までの値を持つ絶対値式[注]
- 相対値式

注 アセンブラでは，16 ビットを越えるかどうかのチェックは行われません。生成された機械語命令では，下位 16 ビットを用いて演算が行われます。

[機能]

- “divhu reg1, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定したレジスタの下位ハーフワード・データの値で符号なしの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。
- “divhu imm, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定した絶対値式，または相対値式の下位ハーフワード・データの値で符号なしの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。

[詳細説明]

- “divhu reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の divhu 命令が 1 つ生成されます。
- “divhu imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。[注]

(a) 0

`divhu  0, reg2, reg3`	`divhu  r0, reg2, reg3`

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

`divhu  imm5, reg2, reg3`	`mov  imm5, r1` `divhu  r1, reg2, reg3`

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`divhu  imm16, reg2, reg3`	`movea  imm16, r0, r1` `divhu  r1, reg2, reg3`

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`divhu  imm, reg2, reg3`	`movhi  HIGHW(imm), r0, r1` `divhu  r1, reg2, reg3`

上記以外の場合

divhu imm, reg2, reg3	mov imm, r1 divhu r1, reg2, reg3

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

divhu $label, reg2, reg3	movea $label, r0, r1 divhu r1, reg2, reg3

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義に持たないラベルの $label を持つ相対値式

divhu #label, reg2, reg3	mov #label, r1 divhu r1, reg2, reg3
divhu label, reg2, reg3	mov label, r1 divhu r1, reg2, reg3
divhu $label, reg2, reg3	mov $label, r1 divhu r1, reg2, reg3

注 機械語命令の divhu 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

divu

符号なし除算（ワード）を行います。（Divide Word Unsigned）

[指定形式]

- divu reg1, reg2, reg3
- divu imm, reg2, reg3

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “divu reg1, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定したレジスタ値で符号なしの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。
- “divu imm, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値を，第 1 オペランドに指定した絶対値式，または相対値式の値で符号なしの値として除算し，商を第 2 オペランドに指定したレジスタに，剰余を第 3 オペランドに指定したレジスタに格納します。第 2 オペランドと第 3 オペランドが同じレジスタの場合，レジスタには剰余を格納します。

[詳細説明]

- “divu reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の divu 命令が 1 つ生成されます。
- “divu imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。[注]

(a) 0

`divu 0, reg2, reg3`	`divu r0, reg2, reg3`

(b) 0 以外の -16 ～＋ 15 の範囲の絶対値式

`divu imm5, reg2, reg3`	`mov imm5, r1` `divu r1, reg2, reg3`

(c) -16 ～＋ 15 の範囲を越え，-32,768 ～ +32,767 の範囲の絶対値式

`divu imm16, reg2, reg3`	`movea imm16, r0, r1` `divu r1, reg2, reg3`

(d) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`divu imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `divu r1, reg2, reg3`

上記以外の場合

`divu imm, reg2, reg3`	`mov imm, r1` `divu r1, reg2, reg3`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`divu  $label, reg2, reg3`	`movea  $label, r0, r1` `divu  r1, reg2, reg3`

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義に持たないラベルの $label を持つ相対値式

`divu  #label, reg2, reg3`	`mov  #label, r1` `divu  r1, reg2, reg3`
`divu  label, reg2, reg3`	`mov  label, r1` `divu  r1, reg2, reg3`
`divu  $label, reg2, reg3`	`mov  $label, r1` `divu  r1, reg2, reg3`

注 機械語命令の divu 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

divq

（符号付き）ワード・データの除算（可変ステップ）を行います。（Divide Word Quickly）

[指定形式]

- divq reg1, reg2, reg3

[機能]

汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し，その商を汎用レジスタ reg2 に，余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。
reg1，reg2 の値から除算に必要となる最小なステップ数を判断して，演算を実行します。
ゼロで割ったときは，オーバフローを生じ，OV フラグ以外の演算結果は不定となります。

[詳細説明]

- アセンブラでは，機械語命令の divq 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	オーバーフローが起こった場合 1，そうでない場合 0
S	演算結果の商が負になった場合 1，そうでない場合 0
Z	演算結果の商が 0 になった場合 1，そうでない場合 0
SAT	—

divqu

（符号なし）ワード・データの除算（可変ステップ）を行います。（Divide Word Unsigned Quickly）

[指定形式]

- divqu reg1, reg2, reg3

[機能]

汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 のワード・データで除算し，その商を汎用レジスタ reg2 に，余りを汎用レジスタ reg3 に格納します。汎用レジスタ reg1 は影響を受けません。
reg1，reg2 の値から除算に必要となる最小なステップ数を判断して，演算を実行します。
ゼロで割ったときは，オーバフローを生じ，OV フラグ以外の演算結果は不定となります。

[詳細説明]

- アセンブラでは，機械語命令の divqu 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	オーバーフローが起こった場合 1，そうでない場合 0
S	演算結果の商のワード・データの MSB が 1 になった場合 1，そうでない場合 0
Z	演算結果の商が 0 になった場合 1，そうでない場合 0
SAT	—

cmp

比較を行います。（Compare）

［指定形式］

- cmp reg1, reg2
- cmp imm, reg2

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

- “cmp reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と第 2 オペランドに指定したレジスタ値を比較し，結果をフラグに示します。なお，比較は，第 2 オペランドに指定したレジスタ値から第 1 オペランドに指定したレジスタ値を減算することにより行われます。
- “cmp imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と第 2 オペランドに指定したレジスタ値を比較し，結果をフラグに示します。なお，比較は，第 2 オペランドに指定したレジスタ値から第 1 オペランドに指定した値を減算することにより行われます。

［詳細説明］

- “cmp reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の cmp 命令が 1 つ生成されます。
- “cmp imm, reg2” の形式の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の cmp 命令[注]が 1 つ生成されます。

(a) -16 ～ +15 の範囲の絶対値式

`cmp imm5, reg`	`cmp imm5, reg`

注 機械語命令の cmp 命令は，第 1 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “cmp imm, reg2” の形式の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`cmp imm16, reg`	`movea imm16, r0, r1` `cmp r1, reg`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`cmp imm, reg`	`mov imm, r1` `cmp r1, reg`

(c) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(d) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

［フラグ］

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

mov

データの転送を行います。（Move）

［指定形式］

- mov reg1, reg2
- mov imm, reg2

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

- “mov reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値を，第 2 オペランドに指定したレジスタに格納します。
- “mov imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値を，第 2 オペランドに指定したレジスタに格納します。

［詳細説明］

- “mov reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の mov 命令が 1 つ生成されます。
- “mov imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の mov 命令[注]が 1 つ生成されます。

(a) -16 ～ +15 の範囲の絶対値式

`mov imm5, reg`	`mov imm5, reg`

注 機械語命令の mov 命令は，第 1 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “mov imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -16 ～ +15 の範囲を越え -32768 ～ +32767 の範囲の絶対値式

`mov imm16, reg`	`movea imm16, r0, reg`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`mov imm, reg`	`movhi HIGHW(imm), r0, reg`

上記以外の場合[注]

`mov imm, reg`	`mov imm, reg`

注 16 ビット長の mov 命令を，48 ビット長の mov 命令に置き換えます。

(c) !label，または %label を持つ相対値式，および sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`mov !label, reg`	`movea !label, r0, reg`
`mov %label, reg`	`movea %label, r0, reg`
`mov $label, reg`	`movea $label, r0, reg`

(d) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式[注]

`mov #label, reg`	`mov #label, reg`
`mov label, reg`	`mov label, reg`
`mov $label, reg`	`mov $label, reg`

注 16 ビット長の mov 命令を，48 ビット長の mov 命令に置き換えます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- “mov imm, reg2” の命令に対し，第 1 オペランドに -16 ～ +15 の範囲の絶対値式を指定し，かつ第 2 オペランドに r0 を指定した場合，または “mov reg1, reg2” の第 2 オペランドに r0 を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

- “mov imm, reg2” の命令に対し，第 1 オペランドに -32768 ～ 32767 の範囲を越える絶対値式，#label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式を指定した場合，アセンブラでは命令展開が行われ，16bit 長の mov 命令を 48bit 長の mov 命令に置き換えられます。疑似命令 .option nomacro の指定により命令展開を抑止している場合にこの命令展開が発生すると，次のメッセージが出力され，アセンブルが中止されます。

E0550249 : 構成に誤りがあります。

movea

実効アドレスの転送を行います。（Move Effective Address）

［指定形式］

- movea imm, reg1, reg2

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

［機能］

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値を，第 2 オペランドに指定したレジスタ値と加算し，結果を第 3 オペランドに指定したレジスタに格納します。

［詳細説明］

- imm に次のものを指定した場合，アセンブラでは，機械語命令の movea 命令[注]が 1 つ生成されます。
- reg1 に r0 を指定した場合，アセンブラでは，mov imm, reg2 を指定したものとして扱います。

(a) -32768 ～ +32767 の範囲の絶対値式

movea imm16, reg1, reg2	movea imm16, reg1, reg2

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

movea $label, reg1, reg2	movea $label, reg1, reg2

(c) !label，または %label を持つ相対値式

movea !label, reg1, reg2	movea !label, reg1, reg2
movea %label, reg1, reg2	movea %label, reg1, reg2

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

movea imm16, reg1, reg2	movea imm16, reg1, reg2

注 機械語命令の movea 命令は，第 1 オペランドに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

movea imm, reg1, reg2	movhi HIGHW(imm), reg1, reg2

上記以外の場合

movea imm, reg1, reg2	movhi HIGHW1(imm), reg1, r1 movea LOWW(imm), r1, reg2

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`movea #label, reg1, reg2`	`movhi HIGHW1(#label), reg1, r1` `movea LOWW(#label), r1, reg2`
`movea label, reg1, reg2`	`movhi HIGHW1(label), reg1, r1` `movea LOWW(label), r1, reg2`
`movea $label, reg1, reg2`	`movhi HIGHW1($label), reg1, r1` `movea LOWW($label), r1, reg2`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- 第 3 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

movhi

上位ハーフワードの転送を行います。(Move High half-word)

[指定形式]

- movhi imm16, reg1, reg2

imm16 に指定可能なものを以下に示します。

- 16 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

[機能]

第 1 オペランドに指定した値を上位 16 ビットの値，0 を下位 16 ビットの値とするワード・データと，第 2 オペランドに指定したレジスタ値を加算し，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

アセンブラでは，機械語命令の movhi 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- imm16 に 0 ～ 65535 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550231 : イミーディエトとして指定された値が指定可能な値の範囲を越えています。

- 第 3 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

  E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

mov32

32 ビット・データの転送を行います。（32bit Move）

［指定形式］

- mov32 imm, reg2

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式[注]

注 相対値式に，HIGHW，LOWW，または HIGHW1 を適用することはできません。

［機能］

第 1 オペランドに指定した絶対値式，または相対値式の値を，第 2 オペランドに指定したレジスタに格納します。

［詳細説明］

アセンブラでは，機械語命令の 48 ビット長 mov 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

cmov

フラグ条件付きデータの転送を行います。（Conditional Move）

［指定形式］

- cmov imm4, reg1, reg2, reg3
- cmov imm4, imm, reg2, reg3
- cmov*cnd* reg1, reg2, reg3
- cmov*cnd* imm, reg2, reg3

imm4 に指定可能なものを以下に示します。

- 4 ビット幅までの値を持つ絶対値式[注]

注　機械語命令の cmov 命令は，第 1 オペランドに 0 ～ 15（0x0 ～ 0xF）の範囲のイミーディエトをとります。

imm に指定可能なものを以下に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

［機能］

- “cmov imm4, reg1, reg2, reg3” の形式
  第 1 オペランドに指定した定数式の値の下位 4 ビットの値で示されるフラグ状態と，現在のフラグ状態を比較し，値が一致した場合は，第 2 オペランドに指定したレジスタ値を，一致しなかった場合は第 3 オペランドに指定したレジスタ値を，第 4 オペランドに指定したレジスタに格納します。
- “cmov imm4, imm, reg2, reg3” の形式
  第 1 オペランドに指定した定数式の値の下位 4 ビットの値で示されるフラグ状態と，現在のフラグ状態を比較し，値が一致した場合は，第 2 オペランドに指定した絶対値式の値を，一致しなかった場合は第 3 オペランドに指定したレジスタ値を，第 4 オペランドに指定したレジスタに格納します。
- “cmov*cnd* reg1, reg2, reg3” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較し，値が一致した場合は，第 1 オペランドに指定したレジスタ値を，一致しなかった場合は第 2 オペランドに指定したレジスタ値を，第 3 オペランドに指定したレジスタに格納します。
- “cmov*cnd* imm, reg2, reg3” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較し，値が一致した場合は，第 1 オペランドに指定した絶対値式の値を，一致しなかった場合は第 2 オペランドに指定したレジスタ値を，第 3 オペランドに指定したレジスタに格納します。

表 5.31　cmov*cnd* 命令

命令	フラグ状態	フラグ状態の意味	命令展開
cmovgt	((S xor OV) or Z) = 0	Greater than（signed）	cmov 0xF
cmovge	(S xor OV) = 0	Greater than or equal（signed）	cmov 0xE
cmovlt	(S xor OV) = 1	Less than（signed）	cmov 0x6
cmovle	((S xor OV) or Z) = 1	Less than or equal（signed）	cmov 0x7
cmovh	(CY or Z) = 0	Higher（Greater than）	cmov 0xB
cmovnl	CY = 0	Not lower（Greater than or equal）	cmov 0x9
cmovl	CY = 1	Lower（Less than）	cmov 0x1

命令	フラグ状態	フラグ状態の意味	命令展開
cmovnh	（CY or Z）= 1	Not higher（Less than or equal）	cmov 0x3
cmove	Z = 1	Equal	cmov 0x2
cmovne	Z = 0	Not equal	cmov 0xA
cmovv	OV = 1	Overflow	cmov 0x0
cmovnv	OV = 0	No overflow	cmov 0x8
cmovn	S = 1	Negative	cmov 0x4
cmovp	S = 0	Positive	cmov 0xC
cmovc	CY = 1	Carry	cmov 0x1
cmovnc	CY = 0	No carry	cmov 0x9
cmovz	Z = 1	Zero	cmov 0x2
cmovnz	Z = 0	Not zero	cmov 0xA
cmovt	always 1	Always 1	cmov 0x5
cmovsa	SAT = 1	Saturated	cmov 0xD

［詳細説明］

- “cmov imm4, reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の cmov 命令[注]が 1 つ生成されます。

注　機械語命令の cmov 命令は，第 2 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “cmov imm4, imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の cmov 命令が 1 つ生成されます。

(a) -16 ～ +15 の範囲の絶対値式

`cmov  imm4, imm5, reg2, reg3`	`cmov  imm4, imm5, reg2, reg3`

- “cmov imm4, imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`cmov  imm4, imm16, reg2, reg3`	`movea  imm16, r0, r1` `cmov  imm4, r1, reg2, reg3`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`cmov  imm4, imm, reg2, reg3`	`movhi  HIGHW(imm), r0, r1` `cmov  imm4, r1, reg2, reg3`

上記以外の場合

`cmov  imm4, imm, reg2, reg3`	`mov  imm, r1` `cmov  imm4, r1, reg2, reg3`

(c) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`cmov imm4, #label, reg2, reg3`	`mov #label, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, label, reg2, reg3`	`mov label, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, $label, reg2, reg3`	`mov $label, r1` `cmov imm4, r1, reg2, reg3`

(d) !label，または %label を持つ相対値式，および sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`cmov imm4, !label, reg2, reg3`	`movea !label, r0, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, %label, reg2, reg3`	`movea %label, r0, r1` `cmov imm4, r1, reg2, reg3`
`cmov imm4, $label, reg2, reg3`	`movea $label, r0, r1` `cmov imm4, r1, reg2, reg3`

- “cmov*cnd* reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，対応する cmov 命令が生成され（「表 5.31 cmovcnd 命令」を参照），“cmov imm4, reg1, reg2, reg3” の形式に展開されます。
- “cmov*cnd* imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，対応する cmov 命令が生成され（「表 5.31 cmovcnd 命令」を参照），“cmov imm, reg2, reg3” の形式に展開されます。

(a) -16 ～ +15 の範囲の絶対値式

- “cmov*cnd* imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`cmov*cnd* imm16, reg2, reg3`	`movea imm16, r0, r1` `cmov*cnd* r1, reg2, reg3`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`cmov*cnd* imm, reg2, reg3`	`movhi HIGHW(imm), r0, r1` `cmov*cnd* r1, reg2, reg3`

上記以外の場合

`cmov*cnd* imm, reg2, reg3`	`mov imm, r1` `cmov*cnd* r1, reg2, reg3`

(c) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`cmov*cnd* #label, reg2, reg3`	`mov #label, r1` `cmov*cnd* r1, reg2, reg3`
`cmov*cnd* label, reg2, reg3`	`mov label, r1` `cmov*cnd* r1, reg2, reg3`
`cmov*cnd* $label, reg2, reg3`	`mov $label, r1` `cmov*cnd* r1, reg2, reg3`

(d) !label，または %label を持つ相対値式，および sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`cmov*cnd* !label, reg2, reg3`	`movea !label, r0, r1` `cmov*cnd* r1, reg2, reg3`
`cmov*cnd* %label, reg2, reg3`	`movea %label, r0, r1` `cmov*cnd* r1, reg2, reg3`
`cmov*cnd* $label, reg2, reg3`	`movea $label, r0, r1` `cmov*cnd* r1, reg2, reg3`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- cmov 命令の imm4 に 4 ビットの範囲を越える定数式を指定した場合，次のメッセージが出力されます。
  なお，4 ビット幅を越える場合は，0xF でマスクした値を用いてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

setf

フラグ条件の設定を行います。（Set Flag Condition）

［指定形式］

- setf imm4, reg
- setf*cnd* reg

imm4 に指定可能なものを以下に示します。

- 4 ビット幅までの値を持つ絶対値式

［機能］

- “setf imm4, reg” の形式
  第 1 オペランドに指定した絶対値式の値の下位 4 ビットの値で示されるフラグ状態と，現在のフラグ状態を比較し，値が一致した場合は 1 を，一致しなかった場合は 0 を第 2 オペランドに指定したレジスタに格納します。
- “setf*cnd* reg” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較し，一致する場合は 1，一致しなかった場合は 0 を第 2 オペランドに指定したレジスタに格納します。

［詳細説明］

- setf 命令に対し，アセンブラでは，機械語命令の setf 命令が 1 つ生成されます。
- setf*cnd* 命令に対し，アセンブラでは，対応する setf 命令が生成され（「表 5.32 setfcnd 命令」を参照），“setf imm4, reg” の形式に展開されます。

表 5.32 setf*cnd* 命令

命令	フラグ状態	フラグ状態の意味	命令展開
setfgt	((S xor OV) or Z) = 0	Greater than（signed）	setf 0xF
setfge	(S xor OV) = 0	Greater than or equal（signed）	setf 0xE
setflt	(S xor OV) = 1	Less than（signed）	setf 0x6
setfle	((S xor OV) or Z) = 1	Less than or equal（signed）	setf 0x7
setfh	(CY or Z) = 0	Higher（Greater than）	setf 0xB
setfnl	CY = 0	Not lower（Greater than or equal）	setf 0x9
setfl	CY = 1	Lower（Less than）	setf 0x1
setfnh	(CY or Z) = 1	Not higher（Less than or equal）	setf 0x3
setfe	Z = 1	Equal	setf 0x2
setfne	Z = 0	Not equal	setf 0xA
setfv	OV = 1	Overflow	setf 0x0
setfnv	OV = 0	No overflow	setf 0x8
setfn	S = 1	Negative	setf 0x4
setfp	S = 0	Positive	setf 0xC
setfc	CY = 1	Carry	setf 0x1
setfnc	CY = 0	No carry	setf 0x9

命令	フラグ状態	フラグ状態の意味	命令展開
setfz	Z = 1	Zero	setf 0x2
setfnz	Z = 0	Not zero	setf 0xA
setft	always 1	Always 1	setf 0x5
setfsa	SAT = 1	Saturated	setf 0xD

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- setf 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定された値の下位 4 ビットが用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

sasf

論理左シフト付きフラグ条件の設定を行います。（Shift And Set Flag Condition）

[指定形式]

- sasf imm4, reg
- sasf*cnd* reg

imm4 に指定可能なものを以下に示します。

- 4 ビット幅までの値を持つ絶対値式

[機能]

- “sasf imm4, reg” の形式
  第 1 オペランドに指定した絶対値式の値の下位 4 ビットの値で示されるフラグ状態と，現在のフラグ状態を比較します。値が一致した場合は，第 2 オペランドで指定したレジスタの内容を左 1 ビット論理シフトした値と 1 とを論理和して，第 2 オペランドに指定したレジスタに格納し，一致しなかった場合は，第 2 オペランドで指定したレジスタの内容を左 1 ビット論理シフトして第 2 オペランドで指定したレジスタに格納します。
- “sasf*cnd* reg” の形式
  cnd 部分の文字列で示されるフラグ状態と現在のフラグの状態を比較します。値が一致した場合は，第 1 オペランドで指定したレジスタの内容を左 1 ビット論理シフトした値と 1 とを論理和して第 1 オペランドで指定したレジスタに格納し，一致しなかった場合は，第 1 オペランドで指定したレジスタの内容を左 1 ビット論理シフトして第 1 オペランドで指定したレジスタに格納します。

[詳細説明]

- sasf 命令に対し，アセンブラでは，機械語命令の sasf 命令が 1 つ生成されます。
- sasf*cnd* 命令に対し，アセンブラでは，対応する sasf 命令が生成され（「表 5.33 sasfcnd 命令」を参照），“sasf imm4, reg” の形式に展開されます。

表 5.33 sasf*cnd* 命令

命令	フラグ状態	フラグ状態の意味	命令展開
sasfgt	((S xor OV) or Z) = 0	Greater than（signed）	sasf 0xF
sasfge	(S xor OV) = 0	Greater than or equal（signed）	sasf 0xE
sasflt	(S xor OV) = 1	Less than（signed）	sasf 0x6
sasfle	((S xor OV) or Z) = 1	Less than or equal（signed）	sasf 0x7
sasfh	(CY or Z) = 0	Higher（Greater than）	sasf 0xB
sasfnl	CY = 0	Not lower（Greater than or equal）	sasf 0x9
sasfl	CY = 1	Lower（Less than）	sasf 0x1
sasfnh	(CY or Z) = 1	Not higher（Less than or equal）	sasf 0x3
sasfe	Z = 1	Equal	sasf 0x2
sasfne	Z = 0	Not equal	sasf 0xA
sasfv	OV = 1	Overflow	sasf 0x0
sasfnv	OV = 0	No overflow	sasf 0x8
sasfn	S = 1	Negative	sasf 0x4

命令	フラグ状態	フラグ状態の意味	命令展開
sasfp	S = 0	Positive	sasf 0xC
sasfc	CY = 1	Carry	sasf 0x1
sasfnc	CY = 0	No carry	sasf 0x9
sasfz	Z = 1	Zero	sasf 0x2
sasfnz	Z = 0	Not zero	sasf 0xA
sasft	always 1	Always 1	sasf 0x5
sasfsa	SAT = 1	Saturated	sasf 0xD

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- sasf 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定された値の下位 4 ビットが用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

bins

レジスタ上のビット挿入を行います。(Bitfield Insert)

[指定形式]

- bins reg1, pos, width, reg2

ipos，width に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

[機能]

汎用レジスタ reg1 の下位 width ビットを取り出し，汎用レジスタ reg2 のフィールド，ビット pos ＋ width － 1 からビット pos に格納する命令です。汎用レジスタ reg2 のうち，指定されたフィールド以外の値は影響を受けません。汎用レジスタ reg1 は影響を受けません。

[詳細説明]

- アセンブラでは，機械語命令の bins 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 になった場合 1，そうでない場合 0
Z	演算結果が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- pos，width に，0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

rotl

ローテートを行います。（Rotate Left）

［指定形式］

- rotl imm, reg2, reg3
- rotl reg1, reg2, reg3

imm に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

［機能］

- “rotl imm, reg2, reg3” の形式
  汎用レジスタ reg2 のワード・データを，ワード長までゼロ拡張した 5 ビット・イミーディエトで示されるシフト数分，左へローテート（回転）し，汎用レジスタ reg3 に書き込みます。汎用レジスタ reg2 は影響を受けません。
- “rotl reg1, reg2, reg3” の形式
  汎用レジスタ reg2 のワード・データを汎用レジスタ reg1 の下位 5 ビットで示されるシフト数分，左へローテート（回転）し，汎用レジスタ reg3 に書き込みます。汎用レジスタ reg1，reg2 は影響を受けません。

［詳細説明］

- “rotl imm, reg2, reg3” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の rotl 命令[注]が 1 つ生成されます。

  (a) 0 ～ 31 の範囲の絶対値式

注 機械語命令の rotl 命令は，第 1 オペランドにレジスタまたは 0 ～ 31（0 ～ 0x1F）の範囲のイミーディエトをとります。

- “rotl reg1, reg2, reg3” の形式
  アセンブラでは，機械語命令の rotl 命令が 1 つ生成されます。

［フラグ］

CY	演算結果のビット 0 が 1 になった場合 1，そうでない場合 0，ただしローテート量が 0 のときは 0
OV	0
S	演算結果が負になった場合 1，そうでない場合 0
Z	演算結果が 0 になった場合 1，そうでない場合 0
SAT	—

5.9.3 飽和演算命令

この項では，飽和演算命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.34 飽和演算命令

命令	意味
satadd	飽和加算
satsub	飽和減算
satsubi	飽和減算（イミーディエト）
satsubr	飽和逆減算

satadd

飽和加算を行います。（Saturated Add）

[指定形式]

- satadd reg1, reg2
- satadd imm, reg2
- satadd reg1, reg2, reg3

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “satadd reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値を加算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。
- “satadd imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタの値を加算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。
- “satadd reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値を加算し，結果を第 3 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 3 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。

[詳細説明]

- “satadd reg1, reg2”，および “satadd reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の satadd 命令が 1 つ生成されます。
- “satadd imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，機械語命令の satadd 命令[注]が 1 つ生成されます。

  (a) -16 ～ +15 の範囲の絶対値式

`satadd  imm5, reg`	`satadd  imm5, reg`

注 機械語命令の satadd 命令は，第 1 オペランドにレジスタ，または -16 ～ +15（0xFFFFFFF0 ～ 0xF）の範囲のイミーディエトをとります。

- “satadd imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

  (a) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`satadd  imm16, reg`	`movea  imm16, r0, r1` `satadd  r1, reg`

(b) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`satadd imm, reg`	`mov imm, r1` `satadd r1, reg`

(c) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(d) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

［フラグ］

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	OV = 1 になった場合 1，そうでない場合 －

［注意事項］

- “satadd reg1, reg2”，および “satadd imm, reg2” の形式の命令に対し，第 2 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

satsub

飽和減算を行います。（Saturated Subtract）

[指定形式]

- satsub reg1, reg2
- satsub imm, reg2
- satsub reg1, reg2, reg3

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “satsub reg1, reg2” の形式
  第 2 オペランドに指定したレジスタ値から，第 1 オペランドに指定したレジスタ値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。
- “satsub imm, reg2” の形式
  第 2 オペランドに指定したレジスタ値から，第 1 オペランドに指定した絶対値式，または相対値式の値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。
- “satsub reg1, reg2, reg3” の形式
  第 2 オペランドに指定したレジスタ値から，第 1 オペランドに指定したレジスタ値を減算し，結果を第 3 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 3 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。

[詳細説明]

- “satsub reg1, reg2”，および “satsub reg1, reg2, reg3” の形式の命令に対し，アセンブラでは，機械語命令の satsub 命令が 1 つ生成されます。
- “satsub imm, reg2” の形式で imm に次のもの，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`satsub  0, reg`	`satsub  r0, reg`

(b) -32768 ～ +32767 の範囲の絶対値式

`satsub  imm16, reg`	`satsubi imm16, reg, reg`

(c) -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

satsub imm, reg	mov imm, r1 satsub r1, reg

(d) data/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

satsub $label, reg	satsubi $label, reg, reg

(e) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

satsub #label, reg	mov #label, r1 satsub r1, reg
satsub label, reg	mov label, r1 satsub r1, reg
satsub $label, reg	mov $label, r1 satsub r1, reg

注 機械語命令の satsub 命令は，オペランドにイミーディエトをとりません。

[フラグ]

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	OV = 1 になった場合 1，そうでない場合 ―

[注意事項]

- “satsub reg1, reg2”，および “satsub imm, reg2” の形式の命令に対し，第 2 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

satsubi

飽和減算を行います。（Saturated Subtract Immediate）

[指定形式]

- satsubi imm, reg1, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

[機能]

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用したものの値を，第 2 オペランドに指定したレジスタ値から減算し，結果を第 3 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 3 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。

[詳細説明]

- imm に次のものを指定した場合，アセンブラでは，機械語命令の satsubi 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`satsubi $label, reg1, reg2`	`satsubi $label, reg1, reg2`

(c) !label，または %label を持つ相対値式

`satsubi !label, reg1, reg2`	`satsubi !label, reg1, reg2`
`satsubi %label, reg1, reg2`	`satsubi %label, reg1, reg2`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`satsubi imm16, reg1, reg2`	`satsubi imm16, reg1, reg2`

注 機械語命令の satsubi 命令は，第 1 オペランドに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- imm に次のものを指定した場合，アセンブラでは命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

<1> imm の値の下位 16 ビットがすべて 0 の場合

`satsubi imm, reg1, reg2`	`movhi  HIGHW(imm), r0, reg2` `satsubr reg1, reg2`

<2> 上記以外

`satsubi imm, reg1, reg2`	`mov  imm, reg2` `satsubr reg1, reg2`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`satsubi #label, reg1, reg2`	`mov  #label, reg2` `satsubr reg1, reg2`
`satsubi label, reg1, reg2`	`mov  label, reg2` `satsubr reg1, reg2`
`satsubi $label, reg1, reg2`	`mov  $label, reg2` `satsubr reg1, reg2`

［フラグ］

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	OV = 1 になった場合 1，そうでない場合 －

［注意事項］

- 第 3 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

satsubr

飽和逆減算を行います。（Saturated Subtract Reverse）

[指定形式]

- satsubr reg1, reg2
- satsubr imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “satsubr reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値から，第 2 オペランドに指定したレジスタ値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。
- “satsubr imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値から，第 2 オペランドに指定したレジスタの値を減算し，結果を第 2 オペランドに指定したレジスタに格納します。ただし，結果が正の最大値 0x7FFFFFFF を越えた場合は 0x7FFFFFFF を，負の最大値 0x80000000 を越えた場合は 0x80000000 を第 2 オペランドに指定したレジスタに格納し，SAT フラグに 1 を設定します。

[詳細説明]

- “satsubr reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の satsubr 命令が 1 つ生成されます。
- “satsubr imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`satsubr 0, reg`	`satsubr r0, reg`

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

`satsubr imm5, reg`	`mov  imm5, r1` `satsubr r1, reg`

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`satsubr imm16, reg`	`movea  imm16, r0, r1` `satsubr r1, reg`

(d) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`satsubr imm, reg`	`mov  imm, r1` `satsubr r1, reg`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の satsubr 命令は，オペランドにイミーディエトをとりません。

[フラグ]

CY	MSB（Most Significant Bit）へのボローが生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	OV = 1 になった場合 1，そうでない場合 －

[注意事項]

- 第 2 オペランドに r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

5.9.4 論理演算命令

この項では，論理演算命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.35 論理演算命令

命令	意味
or	論理和
ori	論理和（イミディエト）
xor	排他的論理和
xori	排他的論理和（イミーディエト）
and	論理積
andi	論理積（イミーディエト）
not	論理否定（1 の補数をとる）
shr	論理右シフト
sar	算術右シフト
shl	論理左シフト
sxb	バイト・データの符号拡張
sxh	2 バイト・データの符号拡張
zxb	バイト・データのゼロ拡張
zxh	2 バイト・データのゼロ拡張
bsh	ハーフワード・データのバイト・スワップ
bsw	ワード・データのバイト・スワップ
hsh	ハーフワード・データのハーフワード・スワップ
hsw	ワード・データのハーフワード・スワップ
tst	テスト
sch0l	MSB 側からのビット（0）検索
sch0r	LSB 側からのビット（0）検索
sch1l	MSB 側からのビット（1）検索
sch1r	LSB 側からのビット（1）検索

or

論理和を行います。（Or）

[指定形式]

- or reg1, reg2
- or imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “or reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値の論理和をとり，結果を第 2 オペランドに指定したレジスタに格納します。
- “or imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタ値の論理和をとり，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “or reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の or 命令が 1 つ生成されます。
- “or imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`or 0, reg`	`or r0, reg`

(b) 1 ～ 65535 の範囲の絶対値式

`or imm16, reg`	`ori imm16, reg, reg`

(c) -16 ～ -1 の範囲の絶対値式

`or imm5, reg`	`mov imm5, r1` `or r1, reg`

(d) -32768 ～ -17 の範囲の絶対値式

`or imm16, reg`	`movea imm16, r0, r1` `or r1, reg`

(e) 上記の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`or imm, reg`	`mov imm, r1` `or r1, reg`

(f) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(g) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の or 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

ori

論理和を行います。（Or Immediate）

[指定形式]

- ori imm, reg1, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

[機能]

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値と，第 2 オペランドに指定したレジスタ値の論理和をとり，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

- imm に次のものを指定した場合，アセンブラでは，機械語命令の ori 命令[注 1] が 1 つ生成されます。

(a) 0 ～ 65535 の範囲の絶対値式

`ori imm16, reg1, reg2`	`ori imm16, reg1, reg2`

(b) !label，または %label を持つ相対値式[注 2]

`ori !label, reg1, reg2`	`ori !label, reg1, reg2`
`ori %label, reg1, reg2`	`ori %label, reg1, reg2`

(c) HIGHW，LOWW，または HIGHW1 を適用したもの

`ori imm16, reg1, reg2`	`ori imm16, reg1, reg2`

注 1. 機械語命令の ori 命令は，第 1 オペランドに 0 ～ 65535（0 ～ 0xFFFF）のイミーディエトをとります。

注 2. !label，$label は，-32768 ～ 32767 の範囲を指定でき，即値（ゼロ拡張）として取り扱います。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -16 ～ -1 の範囲の絶対値式
reg2 が r0 の場合

`ori imm5, reg1, r0`	`mov imm5, r1` `or reg1, r1`

上記以外の場合

`ori imm5, reg1, reg2`	`mov imm5, reg2` `or reg1, reg2`

(b) -32768 ～ -17 の範囲の絶対値式
reg2 が r0 の場合

ori imm16, reg1, r0	movea imm16, r0, r1 or reg1, r1

上記以外の場合

ori imm16, reg1, reg2	movea imm16, r0, reg2 or reg1, reg2

(c) 上記の範囲を越える絶対値式

<1> imm の値の下位 16 ビットがすべて 0 の場合
reg2 が r0 の場合

ori imm, reg1, r0	movhi HIGHW(imm), r0, r1 or reg1, r1

上記以外の場合

ori imm, reg1, reg2	movhi HIGHW(imm), r0, reg2 or reg1, reg2

<2> 上記以外の場合
reg2 が r0 の場合

ori imm, reg1, r0	mov imm, r1 or reg1, r1

上記以外の場合

ori imm, reg1, reg2	mov imm, reg2 or reg1, reg2

(d) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式
reg2 が r0 の場合

ori $label, reg1, r0	movea $label, r0, r1 or reg1, r1

上記以外の場合

ori $label, reg1, reg2	movea $label, r0, reg2 or reg1, reg2

(e) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式
reg2 が r0 の場合

ori #label, reg1, r0	mov #label, r1 or reg1, r1
ori label, reg1, r0	mov label, r1 or reg1, r1
ori $label, reg1, r0	mov $label, r1 or reg1, r1

上記以外の場合

`ori #label, reg1, reg2`	`mov #label, reg2` `or reg1, reg2`
`ori label, reg1, reg2`	`mov label, reg2` `or reg1, reg2`
`ori $label, reg1, reg2`	`mov $label, reg2` `or reg1, reg2`

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

xor

排他的論理和を行います。（Exclusive Or）

[指定形式]

- xor reg1, reg2
- xor imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “xor reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値の排他的論理和をとり，結果を第 2 オペランドに指定したレジスタに格納します。
- “xor imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタ値の排他的論理和をとり，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “xor reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の xor 命令が 1 つ生成されます。
- “xor imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`xor 0, reg`	`xor r0, reg`

(b) 1 ～ 65535 の範囲の絶対値式

`xor imm16, reg`	`xori imm16, reg, reg`

(c) -16 ～ -1 の範囲の絶対値式

`xor imm5, reg`	`mov imm5, r1` `xor r1, reg`

(d) -32768 ～ -17 の範囲の絶対値式

`xor imm16, reg`	`movea imm16, r0, r1` `xor r1, reg`

(e) 上記の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`xor imm, reg`	`mov imm, r1` `xor r1, reg`

(f) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(g) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の xor 命令は，オペランドにイミーディエトをとります。

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

xori

排他的論理和を行います。（Exclusive Or Immediate）

［指定形式］

- xori imm, reg1, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

［機能］

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値と，第 2 オペランドに指定したレジスタ値の排他的論理和をとり，結果を第 3 オペランドに指定したレジスタに格納します。

［詳細説明］

- imm に次のものを指定した場合，アセンブラでは，機械語命令の xori 命令[注 1] が 1 つ生成されます。

(a) 0 ～ 65535 の範囲の絶対値式

xori imm16, reg1, reg2	xori imm16, reg1, reg2

(b) !label，または %label を持つ相対値式[注 2]

xori !label, reg1, reg2	xori !label, reg1, reg2
xori %label, reg1, reg2	xori %label, reg1, reg2

(c) HIGHW，LOWW，または HIGHW1 を適用したもの

xori imm16, reg1, reg2	xori imm16, reg1, reg2

注 1. 機械語命令の xori 命令は，第 1 オペランドに 0 ～ 65535（0 ～ 0xFFFF）のイミーディエトをとります。

注 2. !label，$label は，-32768 ～ 32767 の範囲を指定でき，即値（ゼロ拡張）として取り扱います。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -16 ～ -1 の範囲の絶対値式
reg2 が r0 の場合

xori imm5, reg1, r0	mov imm5, r1 xor reg1, r1

上記以外の場合

xori imm5, reg1, reg2	mov imm5, reg2 xor reg1, reg2

(b) -32768 ～ -17 の範囲の絶対値式
reg2 が r0 の場合

xori imm16, reg1, r0	movea imm16, r0, r1 xor reg1, r1

上記以外の場合

xori imm16, reg1, reg2	movea imm16, r0, reg2 xor reg1, reg2

(c) 上記の範囲を越える絶対値式

<1> imm の値の下位 16 ビットがすべて 0 の場合
reg2 が r0 の場合

xori imm, reg1, r0	movhi HIGHW(imm), r0, r1 xor reg1, r1

上記以外の場合

xori imm, reg1, reg2	movhi HIGHW(imm), r0, reg2 xor reg1, reg2

<2> 上記以外の場合
reg2 が r0 の場合

xori imm, reg1, r0	mov imm, r1 xor reg1, r1

上記以外の場合

xori imm, reg1, reg2	mov imm, reg2 xor reg1, reg2

(d) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式
reg2 が r0 の場合

xori $label, reg1, r0	movea $label, r0, r1 xor reg1, r1

上記以外の場合

xori $label, reg1, reg2	movea $label, r0, reg2 xor reg1, reg2

(e) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式
reg2 が r0 の場合

xori #label, reg1, r0	mov #label, r1 xor reg1, r1
xori label, reg1, r0	mov label, r1 xor reg1, r1
xori $label, reg1, r0	mov $label, r1 xor reg1, r1

上記以外の場合

`xori #label, reg1, reg2`	`mov #label, reg2` `xor reg1, reg2`
`xori label, reg1, reg2`	`mov label, reg2` `xor reg1, reg2`
`xori $label, reg1, reg2`	`mov $label, reg2` `xor reg1, reg2`

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

and

論理積を行います。（And）

[指定形式]

- and reg1, reg2
- and imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “and reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値と，第 2 オペランドに指定したレジスタ値の論理積をとり，結果を第 2 オペランドに指定したレジスタに格納します。
- “and imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，第 2 オペランドに指定したレジスタ値の論理積をとり，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “and reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の and 命令が 1 つ生成されます。
- “and imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`and 0, reg`	`and r0, reg`

(b) 1 ～ 65535 の範囲の絶対値式

`and imm16, reg`	`andi imm16, reg, reg`

(c) -16 ～ -1 の範囲の絶対値式

`and imm5, reg`	`mov imm5, r1` `and r1, reg`

(d) -32768 ～ -17 の範囲の絶対値式

`and imm16, reg`	`movea imm16, r0, r1` `and r1, reg`

(e) 上記の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`and imm, reg`	`mov imm, r1` `and r1, reg`

(f) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(g) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の and 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

andi

論理積を行います。（And Immediate）

［指定形式］

- andi imm, reg1, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

［機能］

第 1 オペランドに指定した絶対値式，相対値式，あるいは，HIGHW，LOWW，または HIGHW1 を適用した値と，第 2 オペランドに指定したレジスタ値の論理積をとり，結果を第 3 オペランドに指定したレジスタに格納します。

［詳細説明］

- imm に次のものを指定した場合，アセンブラでは，機械語命令の andi 命令[注 1] が 1 つ生成されます。

(a) 0 ～ 65535 の範囲の絶対値式

`andi imm16, reg1, reg2`	`andi imm16, reg1, reg2`

(b) !label，または %label を持つ相対値式[注 2]

`andi !label, reg1, reg2`	`andi !label, reg1, reg2`
`andi %label, reg1, reg2`	`andi %label, reg1, reg2`

(c) HIGHW，LOWW，または HIGHW1 を適用したもの

`andi imm16, reg1, reg2`	`andi imm16, reg1, reg2`

注 1. 機械語命令の andi 命令は，第 1 オペランドに 0 ～ 65535（0 ～ 0xFFFF）のイミーディエトをとります。

注 2. !label，$label は，-32768 ～ 32767 の範囲を指定でき，即値（ゼロ拡張）として取り扱います。

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。

(a) -16 ～ -1 の範囲の絶対値式
reg2 が r0 の場合

`andi imm5, reg1, r0`	`mov imm5, r1` `and reg1, r1`

上記以外の場合

`andi imm5, reg1, reg2`	`mov imm5, reg2` `and reg1, reg2`

(b) -32768 ～ -17 の範囲の絶対値式
reg2 が r0 の場合

`andi imm16, reg1, r0`	`movea imm16, r0, r1` `and reg1, r1`

上記以外の場合

`andi imm16, reg1, reg2`	`movea imm16, r0, reg2` `and reg1, reg2`

(c) 上記の範囲を越える絶対値式

<1> imm の値の下位 16 ビットがすべて 0 の場合
reg2 が r0 の場合

`andi imm, reg1, r0`	`movhi HIGHW(imm), r0, r1` `and reg1, r1`

上記以外の場合

`andi imm, reg1, reg2`	`movhi HIGHW(imm), r0, reg2` `and reg1, reg2`

<2> 上記以外の場合
reg2 が r0 の場合

`andi imm, reg1, r0`	`mov imm, r1` `and reg1, r1`

上記以外の場合

`andi imm, reg1, reg2`	`mov imm, reg2` `and reg1, reg2`

(d) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式
reg2 が r0 の場合

`andi $label, reg1, r0`	`movea $label, r0, r1` `and reg1, r1`

上記以外の場合

`andi $label, reg1, reg2`	`movea $label, r0, reg2` `and reg1, reg2`

(e) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式
reg2 が r0 の場合

`andi #label, reg1, r0`	`mov #label, r1` `and reg1, r1`
`andi label, reg1, r0`	`mov label, r1` `and reg1, r1`
`andi $label, reg1, r0`	`mov $label, r1` `and reg1, r1`

上記以外の場合

`andi #label, reg1, reg2`	`mov #label, reg2` `and reg1, reg2`
`andi label, reg1, reg2`	`mov label, reg2` `and reg1, reg2`
`andi $label, reg1, reg2`	`mov $label, reg2` `and reg1, reg2`

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

not

論理否定（1 の補数をとる）を行います。（Not）

[指定形式]

- not reg1, reg2
- not imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “not reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値の論理否定（1 の補数）をとり，結果を第 2 オペランドに指定したレジスタに格納します。
- “not imm, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値の論理否定（1 の補数）をとり，結果を第 2 オペランドに指定したレジスタに格納します。

[詳細説明]

- “not reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の not 命令が 1 つ生成されます。
- “not imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`not 0, reg`	`not r0, reg`

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

`not imm5, reg`	`mov imm5, r1` `not r1, reg`

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`not imm16, reg`	`movea imm16, r0, r1` `not r1, reg`

(d) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

`not imm, reg`	`movhi HIGHW(imm), r0, r1` `not r1, reg`

上記以外の場合

`not imm, reg`	`mov imm, r1` `not r1, reg`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`not $label, reg`	`movea $label, r0, r1` `not r1, reg`

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`not #label, reg`	`mov #label, r1` `not r1, reg`
`not label, reg`	`mov label, r1` `not r1, reg`
`not $label, reg`	`mov $label, r1` `not r1, reg`

注 機械語命令の not 命令は，オペランドにイミーディエトをとりません。

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

shr

論理右シフトを行います。（Shift Logical Right）

[指定形式]

- shr reg1, reg2
- shr imm5, reg2
- shr reg1, reg2, reg3

imm5 に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

[機能]

- “shr reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を右に論理シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “shr imm5, reg2” の形式
  第 1 オペランドに指定した絶対値式の値で示されるビット数分，第 2 オペランドに指定したレジスタ値を右に論理シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “shr reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を右に論理シフトし，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

アセンブラでは，機械語命令の shr 命令が 1 つ生成されます。

[フラグ]

CY	最後にシフト・アウトしたビットの値が 1 の場合 1，そうでない場合 0 （指定したビット数が 0 の場合 0）
OV	0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- “shr imm5, reg2” の形式で imm5 に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値の下位 5 ビット[注]が用いられてアセンブルが続行されます。

```
W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。
```

注 機械語命令の shr 命令は，第 1 オペランドに 0 ～ 31（0x0 ～ 0x1F）のイミーディエトをとります。

sar

算術右シフトを行います。（Shift Arithmetic Right）

[指定形式]

- sar reg1, reg2
- sar imm5, reg2
- sar reg1, reg2, reg3

imm5 に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

[機能]

- “sar reg1, reg2” の形式
  第 1 オペランドに指定したレジスタの値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を右に算術シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “sar imm5, reg2” の形式
  第 1 オペランドに指定した絶対値式の値で示されるビット数分，第 2 オペランドに指定したレジスタ値を右に算術シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “sar reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を右に算術シフトし，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

アセンブラでは，機械語命令の sar 命令が 1 つ生成されます。

[フラグ]

CY	最後にシフト・アウトしたビットの値が 1 の場合 1，そうでない場合 0 （指定したビット数が 0 の場合 0）
OV	0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- “sar imm5, reg2” の形式で imm5 に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値の下位 5 ビット[注]が用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の sar 命令は，第 1 オペランドに 0 ～ 31（0x0 ～ 0x1F）のイミーディエトをとります。

shl

論理左シフトを行います。（Shift Logical Left）

[指定形式]

- shl reg1, reg2
- shl imm5, reg2
- shl reg1, reg2, reg3

imm5 に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

[機能]

- “shl reg1, reg2” の形式
  第 1 オペランドに指定したレジスタ値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を左に論理シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “shl imm5, reg2” の形式
  第 1 オペランドに指定した絶対値式の値で示されるビット数分，第 2 オペランドに指定したレジスタ値を左に論理シフトし，結果を第 2 オペランドに指定したレジスタに格納します。
- “shl reg1, reg2, reg3” の形式
  第 1 オペランドに指定したレジスタ値の下位 5 ビットで示されるビット数分，第 2 オペランドに指定したレジスタ値を左に論理シフトし，結果を第 3 オペランドに指定したレジスタに格納します。

[詳細説明]

アセンブラでは，機械語命令の shl 命令が 1 つ生成されます。

[フラグ]

CY	最後にシフト・アウトしたビットの値が 1 の場合 1，そうでない場合 0（指定したビット数が 0 の場合 0）
OV	0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- “shl imm5, reg2” の形式で imm5 に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値の下位 5 ビット[注]が用いられてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の shl 命令は，第 1 オペランドに 0 ～ 31（0x0 ～ 0x1F）のイミーディエトをとります。

sxb

バイト・データの符号拡張を行います。（Sign Extend Byte）

［指定形式］

- sxb reg

［機能］

第 1 オペランドに指定したレジスタの下位 1 バイトのデータを，ワード長に符号拡張します。

［詳細説明］

アセンブラでは，機械語命令の sxb 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

sxh

2 バイト・データの符号拡張を行います。（Sign Extend Half-word）

［指定形式］

- sxh reg

［機能］

第 1 オペランドに指定したレジスタの下位 2 バイトのデータを，ワード長に符号拡張します。

［詳細説明］

アセンブラでは，機械語命令の sxh 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

zxb

バイト・データのゼロ拡張を行います。(Zero Extend Byte)

[指定形式]

- zxb reg

[機能]

第 1 オペランドに指定したレジスタの下位 1 バイトのデータを，ワード長にゼロ拡張します。

[詳細説明]

アセンブラでは，機械語命令の zxb 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

zxh

2 バイト・データのゼロ拡張を行います。（Zero Extend Half-word）

［指定形式］

- zxh reg

［機能］

第 1 オペランドに指定したレジスタの下位 2 バイトのデータを，ワード長にゼロ拡張します。

［詳細説明］

アセンブラでは，機械語命令の zxh 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

bsh

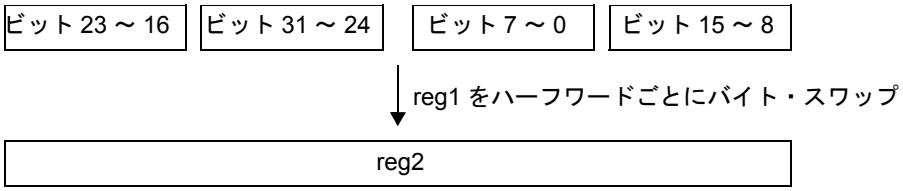
ハーフワード・データのバイト・スワップを行います。（Byte Swap Half-word）

［指定形式］

- bsh reg1, reg2

［機能］

第 1 オペランドに指定したレジスタ値を，ハーフワード単位でバイト・スワップして，第 2 オペランドに指定したレジスタに格納します。



［詳細説明］

アセンブラでは，機械語命令の bsh 命令が 1 つ生成されます。

［フラグ］

CY	レジスタの下位ハーフワード中の 1 つ以上のバイトが 0 の場合 1，そうでない場合 0
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	演算結果の下位ハーフ・ワード・データが 0 になった場合 1，そうでない場合 0
SAT	—

bsw

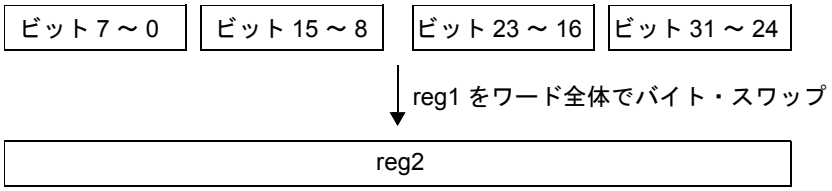
ワード・データのバイト・スワップを行います。（Byte Swap Word）

［指定形式］

- bsw reg1, reg2

［機能］

第 1 オペランドに指定したレジスタ値を，バイト・スワップして，第 2 オペランドに指定したレジスタに格納します。



［詳細説明］

アセンブラでは，機械語命令の bsw 命令が 1 つ生成されます。

［フラグ］

CY	レジスタのワード中の 1 つ以上のバイトが 0 の場合 1，そうでない場合 0
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	演算結果のワード・データが 0 になった場合 1，そうでない場合 0
SAT	—

hsh

ハーフワード・データのハーフワード・スワップを行います。（Half-word Swap Half-word）

［指定形式］

- hsh reg1, reg2

［機能］

第 1 オペランドに指定したレジスタ値を第 2 オペランドに指定したレジスタに格納し，フラグの判定結果を PSW レジスタに格納します。

［詳細説明］

アセンブラでは，機械語命令の hsh 命令が 1 つ生成されます。

［フラグ］

CY	演算結果の下位ハーフワードが 0 の場合 1，そうでない場合 0
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	演算結果の下位ハーフワードが 0 の場合 1，そうでない場合 0
SAT	—

hsw

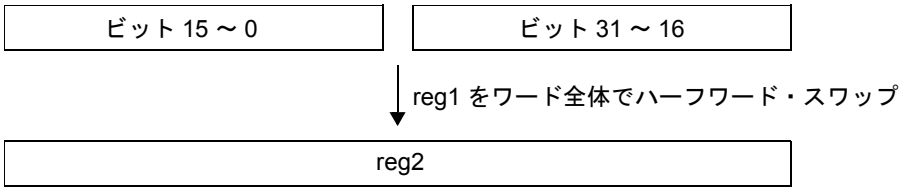
ワード・データのハーフワード・スワップを行います。（Half-word Swap Word）

［指定形式］

- hsw reg1, reg2

［機能］

第 1 オペランドに指定したレジスタ値を，ハーフワード・スワップして，第 2 オペランドに指定したレジスタに格納します。



［詳細説明］

アセンブラでは，機械語命令の hsw 命令が 1 つ生成されます。

［フラグ］

CY	演算結果のワード・データ中に 0 のハーフワードが 1 つ以上含まれる場合 1，そうでない場合 0
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	演算結果のワード・データが 0 になった場合 1，そうでない場合 0
SAT	—

tst

テストを行います。（Test）

[指定形式]

- tst reg1, reg2
- tst imm, reg2

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式

[機能]

- “tst reg1, reg2” の形式
  第 2 オペランドに指定したレジスタ値と，第 1 オペランドに指定したレジスタ値の論理積をとり，結果は格納せず，フラグのみを設定します。
- “tst imm, reg2” の形式
  第 2 オペランドに指定したレジスタ値と，第 1 オペランドに指定した絶対値式，または相対値式の値の論理積をとり，結果は格納せず，フラグのみを設定します。

[詳細説明]

- “tst reg1, reg2” の形式の命令に対し，アセンブラでは，機械語命令の tst 命令が 1 つ生成されます。
- “tst imm, reg2” の形式で imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，1 つ，または複数個の機械語命令が生成されます。注

(a) 0

`tst 0, reg`	`tst r0, reg`

(b) 0 以外の -16 ～ +15 の範囲の絶対値式

`tst imm5, reg`	`mov imm5, r1` `tst r1, reg`

(c) -16 ～ +15 の範囲を越え，-32768 ～ +32767 の範囲の絶対値式

`tst imm16, reg`	`movea imm16, r0, r1` `tst r1, reg`

(d) imm に -32768 ～ +32767 の範囲を越える絶対値式
imm の値の下位 16 ビットがすべて 0 の場合

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

上記以外の場合

`tst imm, reg`	`mov imm, r1` `tst r1, reg`

(e) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

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

(f) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

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

注 機械語命令の tst 命令は，オペランドにイミーディエトをとります。

［フラグ］

CY	—
OV	0
S	演算結果のワード・データの MSB が 1 の場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

sch0l

MSB 側からのビット（0）検索を行います。（SearchZero from Left）

[指定形式]

- sch0l reg1, reg2

[機能]

第 1 オペランドに指定したレジスタのワード・データを左側（MSB 側）から検索し，最初にビット（0）が見つかった位置を第 2 オペランドに指定したレジスタに 16 進数で格納します（たとえば，第 1 オペランドに指定したレジスタのビット 31 が 0 の場合は，第 2 オペランドに指定したレジスタに 01H を格納します）。

ビット（0）が見つからなかった場合は，第 2 オペランドに指定したレジスタに 0 を書き込み，同時に Z フラグをセット（1）します。最後にビット（0）が見つかった場合は，CY フラグをセット（1）します。

[詳細説明]

アセンブラでは，機械語命令の sch0l 命令が 1 つ生成されます。

[フラグ]

CY	最後にビット（0）が見つかった場合 1，そうでない場合 0
OV	0
S	0
Z	ビット（0）が見つからなかった場合 1，そうでない場合 0
SAT	—

sch0r

LSB 側からのビット（0）検索を行います。（Search Zero from Right）

[指定形式]

- sch0r reg1, reg2

[機能]

第 1 オペランドに指定したレジスタのワード・データを右側（LSB 側）から検索し，最初にビット（0）が見つかった位置を第 2 オペランドに指定したレジスタに 16 進数で格納します（たとえば，第 1 オペランドに指定したレジスタのビット 0 が 0 の場合は，第 2 オペランドに指定したレジスタに 01H を格納します）。

ビット（0）が見つからなかった場合は，第 2 オペランドに指定したレジスタに 0 を書き込み，同時に Z フラグをセット（1）します。最後にビット（0）が見つかった場合は，CY フラグをセット（1）します。

[詳細説明]

アセンブラでは，機械語命令の sch0r 命令が 1 つ生成されます。

[フラグ]

CY	最後にビット（0）が見つかった場合 1，そうでない場合 0
OV	0
S	0
Z	ビット（0）が見つからなかった場合 1，そうでない場合 0
SAT	—

sch1l

MSB 側からのビット（1）検索を行います。（Search One from Left）

[指定形式]

- sch1l reg1, reg2

[機能]

第 1 オペランドに指定したレジスタのワード・データを左側（MSB 側）から検索し，最初にビット（1）が見つかった位置を第 2 オペランドに指定したレジスタに 16 進数で書き込みます（たとえば，第 1 オペランドに指定したレジスタのビット 31 が 1 の場合は，第 2 オペランドに指定したレジスタに 01H を格納します）。

ビット（1）が見つからなかった場合は，第 2 オペランドに指定したレジスタに 0 を書き込み，同時に Z フラグをセット（1）します。最後にビット（1）が見つかった場合は，CY フラグをセット（1）します。

[詳細説明]

アセンブラでは，機械語命令の sch1l 命令が 1 つ生成されます。

[フラグ]

CY	最後にビット（1）が見つかった場合 1，そうでない場合 0
OV	0
S	0
Z	ビット（1）が見つからなかった場合 1，そうでない場合 0
SAT	—

sch1r

LSB 側からのビット（1）検索を行います。（Search One from Right）

[指定形式]

- sch1r reg1, reg2

[機能]

第 1 オペランドに指定したレジスタのワード・データを右側（LSB 側）から検索し，最初にビット（1）が見つかった位置を第 2 オペランドに指定したレジスタに 16 進数で格納します（たとえば，第 1 オペランドに指定したレジスタのビット 0 が 1 の場合は，第 2 オペランドに指定したレジスタに 01H を格納します）。

ビット（1）が見つからなかった場合は，第 2 オペランドに指定したレジスタに 0 を書き込み，同時に Z フラグをセット（1）します。最後にビット（1）が見つかった場合は，CY フラグをセット（1）します。

[詳細説明]

アセンブラでは，機械語命令の sch1r 命令が 1 つ生成されます。

[フラグ]

CY	最後にビット（1）が見つかった場合 1，そうでない場合 0
OV	0
S	0
Z	ビット（1）が見つからなかった場合 1，そうでない場合 0
SAT	—

5.9.5 分岐命令

この項では，分岐命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.36 分岐命令

命令	意味
jmp	無条件分岐
jmp32	無条件分岐
jr	無条件分岐（PC 相対）
jr22	無条件分岐（PC 相対）
jr32	無条件分岐（PC 相対）
jcnd	条件分岐
jcnd9	条件分岐
jcnd17	条件分岐
jarl	ジャンプ・アンド・レジスタ・リンク
jarl22	ジャンプ・アンド・レジスタ・リンク
jarl32	ジャンプ・アンド・レジスタ・リンク

jmp

無条件分岐を行います。（Jump）

[指定形式]

- jmp [reg]
- jmp addr
- jmp disp32[reg]

addr に指定できるものを次に示します。

- ラベルの絶対アドレス参照を持つ相対値式

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの絶対アドレス参照，およびセクション内オフセット参照を持つ相対値式

[機能]

- “jmp [reg]” の形式
  オペランドに指定したレジスタ値が示すアドレスに制御を移します。
- “jmp addr” の形式
  オペランドに指定した相対値式の値が示すアドレスに制御を移します。
- “jmp disp32[reg]” の形式
  オペランドに指定したディスプレースメントとレジスタの内容を加算して得たアドレスに制御を移します。

[詳細説明]

- “jmp [reg]” の形式の命令に対し，アセンブラでは，機械語命令の jmp 命令が 1 つ生成されます。
- “jmp addr” の形式の命令に対し，RH850 動作時には，アセンブラでは，機械語命令の jmp 命令（6 バイト長命令）が 1 つ生成されます。
- “jmp disp32[reg]” の形式の命令に対し，アセンブラでは，機械語命令の jmp 命令（6 バイト長命令）が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- “jmp addr” の形式において，addr にラベルの絶対アドレス参照を持つ相対値式以外のものを指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550224 : jmp 命令に対し、絶対アドレス参照（string）以外のものが指定されています。

jmp32

無条件分岐を行います。（Jump）

[指定形式]

- jmp32 disp32[reg]
- jmp32 addr

addr に指定できるものを次に示します。

- ラベルの絶対アドレス参照を持つ相対値式

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの絶対アドレス参照，およびセクション内オフセット参照を持つ相対値式

[機能]

- “jmp32 disp32[reg]” の形式
  オペランドに指定したディスプレースメントとレジスタの内容を加算して得たアドレスに制御を移します。
- “jmp32 addr” の形式
  オペランドに指定した相対値式の値が示すアドレスに制御を移します。

[詳細説明]

アセンブラでは，機械語命令の jmp 命令（6 バイト長命令）が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- “jmp32 addr” の形式において，addr にラベルの絶対アドレス参照を持つ相対値式以外のものを指定した場合，次のメッセージが出力され，アセンブルが中止されます。

```
E0550224 : jmp 命令に対し、絶対アドレス参照（string）以外のものが指定されています。
```

jr

無条件分岐（PC 相対）を行います。（Jump Relative）

[指定形式]

- jr disp22
- jr disp32

disp22 に指定できるものを次に示します。

- 22 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

[機能]

- “jr disp22” の形式
  オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。
- “jr disp32” の形式
  オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。

[詳細説明]

- “jr disp22” の形式の命令に対し，disp22 に次のものを指定した場合，アセンブラで機械語命令の jr 命令[注]が 1 つ生成されます。
  (a) -2097152 ～ +2097151 の範囲の絶対値式
  (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-2097152 ～ +2097151 の範囲の相対値式
  (c) 本命令と同じファイル内に定義を持たないか同じセクションに定義を持たないラベルの PC オフセット参照を持つ相対値式

注 機械語命令の jr は，ディスプレースメントに -2097152 ～ +2097151（0xFE00000 ～ 0x1FFFFF）の範囲のイミーディエトをとります。

- “jr disp32” の形式の命令に対し，アセンブラでは，機械語命令の jr 命令（6 バイト長命令）が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- disp22 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp22/disp32 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550226 : 奇数のディスプレースメントが指定されています。

- アセンブラ・-Xfar_jump オプションを指定しない場合に，disp32 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

jr22

無条件分岐（PC 相対）を行います。（Jump Relative）

[指定形式]

- jr22 disp22

disp22 に指定できるものを次に示します。

- 22 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

[機能]

オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。

[詳細説明]

- disp22 に次のものを指定した場合，アセンブラでは，機械語命令の jr 命令[注]が 1 つ生成されます。
  - (a) -2097152 ～ +2097151 の範囲の絶対値式
  - (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-2097152 ～ +2097151 の範囲の相対値式
  - (c) 本命令と同じファイル内に定義を持たないか同じセクションに定義を持たないラベルの PC オフセット参照を持つ相対値式

注 機械語命令の jr は，ディスプレースメントに -2097152 ～ +2097151（0xFE00000 ～ 0x1FFFFF）の範囲のイミーディエトをとります。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp22 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp22 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550226 : 奇数のディスプレースメントが指定されています。

jr32

無条件分岐（PC 相対）を行います。（Jump Relative）

[指定形式]

- jr32 disp32

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

[機能]

オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。

[詳細説明]

アセンブラでは，機械語命令の jr 命令（6 バイト長命令）が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp32 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550226 : 奇数のディスプレースメントが指定されています。

jcnd

条件分岐を行います。（Jump on Condition）

［指定形式］

- *jcnd* disp22

disp22 に指定できるものを次に示します。

- 22 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

［機能］

cnd 部分の文字列で示されるフラグ状態（「表 5.37 jcnd 命令」を参照）と，現在のフラグ状態を比較し，一致した場合は，オペランドに指定した絶対値式，または相対値式の値と現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。[注]

注 jbr 以外の *jcnd* 命令に対しては，b*cnd* というニモニックを，jbr 命令に対しては br という機械語命令を用いることもできます（機能に違いはありません）。

表 5.37 *jcnd* 命令

命令	フラグ状態	フラグ状態の意味
jgt	((S xor OV) or Z) = 0	Greater than（signed）
jge	(S xor OV) = 0	Greater than or equal（signed）
jlt	(S xor OV) = 1	Less than（signed）
jle	((S xor OV) or Z) = 1	Less than or equal（signed）
jh	(CY or Z) = 0	Higher（Greater than）
jnl	CY = 0	Not lower（Greater than or equal）
jl	CY = 1	Lower（Less than）
jnh	(CY or Z) = 1	Not higher（Less than or equal）
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	—	Always（無条件）
jsa	SAT = 1	Saturated

[詳細説明]

- disp22 に次のものを指定した場合，アセンブラでは，機械語命令の b*cnd* 命令[注]が 1 つ生成されます。
  (a) -256 ～ +255 の範囲の絶対値式
  (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-256 ～ +255 の範囲の絶対値式

`jcnd disp9`	`bcnd disp9`

注 機械語命令の b*cnd* は，ディスプレースメントに -256 ～ +255（0xFFFFFF00 ～ 0xFF）の範囲のイミーディエトをとります。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp22 に，-2097150 ～ +2097153 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097150 ～ +2097153 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp22 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550226 : 奇数のディスプレースメントが指定されています。

- disp22 に，17 ビット幅の絶対値式を持つ *jcnd* 命令を指定したい場合は，*jcnd*17 命令を指定してください。

*jcnd*9

条件分岐を行います。（Jump on Condition）

［指定形式］

- *jcnd*9 disp9

disp9 に指定できるものを次に示します。

- 9 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

［機能］

cnd 部分の文字列で示されるフラグ状態（「表 5.38 jcnd9 命令」を参照）と，現在のフラグ状態を比較し，一致した場合は，オペランドに指定した絶対値式，または相対値式の値と現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。[注]

注 jbr 以外の *jcnd* 命令に対しては，b*cnd* というニモニックを，jbr 命令に対しては br という機械語命令を用いることもできます（機能に違いはありません）。

表 5.38 *jcnd*9 命令

命令	フラグ状態	フラグ状態の意味
jgt	((S xor OV) or Z) = 0	Greater than（signed）
jge	(S xor OV) = 0	Greater than or equal（signed）
jlt	(S xor OV) = 1	Less than（signed）
jle	((S xor OV) or Z) = 1	Less than or equal（signed）
jh	(CY or Z) = 0	Higher（Greater than）
jnl	CY = 0	Not lower（Greater than or equal）
jl	CY = 1	Lower（Less than）
jnh	(CY or Z) = 1	Not higher（Less than or equal）
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	—	Always（無条件）
jsa	SAT = 1	Saturated

[詳細説明]

- disp9 に次のものを指定した場合，アセンブラでは，機械語命令の b*cnd* 命令が 1 つ生成されます。
  (a) -256 ～ +255 の範囲の絶対値式
  (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-256 ～ +255 の範囲の絶対値式

`jcnd disp9`	`bcnd disp9`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp9 に，-256 ～ +255 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -256 ～ +255 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp9 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550226 : 奇数のディスプレースメントが指定されています。

*jcnd*17

条件分岐を行います。（Jump on Condition）

［指定形式］

- *jcnd*17 disp17

disp17 に指定できるものを次に示します。

- 17 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

［機能］

cnd 部分の文字列で示されるフラグ状態（「表 5.39 jcnd17 命令」を参照）と，現在のフラグ状態を比較し，一致した場合は，オペランドに指定した絶対値式，または相対値式の値と現在のプログラム・カウンタ（PC）の値を加算したアドレスに制御を移します。注

注 jbr 以外の *jcnd* 命令に対しては，b*cnd* というニモニックを，jbr 命令に対しては br という機械語命令を用いることもできます（機能に違いはありません）。

表 5.39 *jcnd*17 命令

命令	フラグ状態	フラグ状態の意味
jgt	((S xor OV) or Z) = 0	Greater than（signed）
jge	(S xor OV) = 0	Greater than or equal（signed）
jlt	(S xor OV) = 1	Less than（signed）
jle	((S xor OV) or Z) = 1	Less than or equal（signed）
jh	(CY or Z) = 0	Higher（Greater than）
jnl	CY = 0	Not lower（Greater than or equal）
jl	CY = 1	Lower（Less than）
jnh	(CY or Z) = 1	Not higher（Less than or equal）
je	Z = 1	Equal
jne	Z = 0	Not equal
jv	OV = 1	Overflow
jnv	OV = 0	No overflow
jn	S = 1	Negative
jp	S = 0	Positive
jc	CY = 1	Carry
jnc	CY = 0	No carry
jz	Z = 1	Zero
jnz	Z = 0	Not zero
jbr	—	Always（無条件）
jsa	SAT = 1	Saturated

[詳細説明]

- disp17 に次のものを指定した場合，アセンブラでは，機械語命令の b*cnd* 命令が 1 つ生成されます。
  (a) -65536 ～ +65535 の範囲の絶対値式
  (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-65536 ～ +65535 の範囲の絶対値式

`jcnd disp17`	`bcnd disp17`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp17 に，-65536 ～ +65535 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -65536 ～ +65535 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp17 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550226 : 奇数のディスプレースメントが指定されています。

jarl

ジャンプ・アンド・レジスタ・リンクを行います。（Jump and Register Link）

［指定形式］

- jarl disp22, reg2
- jarl disp32, reg2
- jarl [reg1], reg3

disp22 に指定できるものを次に示します。

- 22 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

［機能］

- “jarl disp22, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）値を加算したアドレスに制御を移します。なお，戻りアドレスは，第 2 オペランドに指定したレジスタに格納されます。
- “jarl disp32, reg2” の形式
  第 1 オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）値を加算したアドレスに制御を移します。なお，戻りアドレスは，第 2 オペランドに指定したレジスタに格納されます。
- “jarl [reg1], reg3” の形式
  第 1 オペランドに指定したレジスタ値が示すアドレスと，現在のプログラム・カウンタ（PC）値を加算したアドレスに制御を移します。なお，戻りアドレスは，第 2 オペランドに指定したレジスタに格納されます。

［詳細説明］

- “jarl disp22, reg2” の形式の命令に対し，disp22 に次のものを指定した場合，アセンブラでは，機械語命令の jarl 命令[注]が 1 つ生成されます。

  (a) -2097152 ～ +2097151 の範囲の絶対値

  (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-2097152 ～ +2097151 の範囲の相対値式

  (c) 本命令と同じファイル内に定義を持たないか，同じセクションに定義を持たないラベルの PC オフセット参照を持つ相対値式

  注 機械語命令の jarl は，オペランドに -2097152 ～ +2097151（0xFE00000 ～ 0x1FFFFF）の範囲のイミーディエトをとります。

- “jarl disp32, reg2” の形式の命令に対し，アセンブラでは，機械語命令の jarl 命令（6 バイト長命令）が 1 つ生成されます。
- “jarl [reg1], reg3” の形式の命令に対し，アセンブラでは，機械語命令の jarl 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- disp22 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp22/disp32 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550226 : 奇数のディスプレースメントが指定されています。

- アセンブラ・-Xfar_jump オプションを指定しない場合に，disp32 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

  E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- reg3 に r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

  E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

jarl22

ジャンプ・アンド・レジスタ・リンクを行います。（Jump and Register Link）

[指定形式]

- jarl22 disp22, reg1

disp22 に指定できるものを次に示します。

- 22 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

[機能]

第 1 オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）値を加算したアドレスに制御を移します。なお，戻りアドレスは，第 2 オペランドに指定したレジスタに格納されます。

[詳細説明]

- disp22 に次のものを指定した場合，アセンブラでは，機械語命令の jarl 命令[注]が 1 つ生成されます。
  - (a) -2097152 ～ +2097151 の範囲の絶対値
  - (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，-2097152 ～ +2097151 の範囲の相対値式
  - (c) 本命令と同じファイル内に定義を持たないか，同じセクションに定義を持たないラベルの PC オフセット参照を持つ相対値式

注 機械語命令の jarl は，オペランドに -2097152 ～ +2097151（0xFE00000 ～ 0x1FFFFF）の範囲のイミーディエトをとります。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp22 に，-2097152 ～ +2097151 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち -2097152 ～ +2097151 の範囲を越える相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

```
E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。
```

- disp22 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

```
E0550226 : 奇数のディスプレースメントが指定されています。
```

jarl32

ジャンプ・アンド・レジスタ・リンクを行います。（Jump and Register Link）

[指定形式]

- jarl32 disp32, reg2

disp32 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式

[機能]

第 1 オペランドに指定した絶対値式，または相対値式の値と，現在のプログラム・カウンタ（PC）値を加算したアドレスに制御を移します。なお，戻りアドレスは，第 2 オペランドに指定したレジスタに格納されます。

[詳細説明]

アセンブラでは，機械語命令の jarl 命令（6 バイト長命令）が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- disp32 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550226 : 奇数のディスプレースメントが指定されています。

5.9.6 ビット操作命令

この項では，ビット操作命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.40 ビット操作命令

命令	意味
set1	ビット・セット
clr1	ビット・クリア
not1	ビット・ノット
tst1	ビット・テスト

set1

ビット・セットを行います。（Set Bit）

［指定形式］

- set1 bit#3, disp[reg1]
- set1 reg2, [reg1]

disp に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

注意 “set1 reg2, [reg1]” の形式では disp は指定できません。

［機能］

- “set1 bit#3, disp[reg1]” の形式
  第 2 オペランドで指定したアドレスが示すデータの，第 1 オペランドに指定したビットをセットします。指定したビット以外は影響を受けません。
- “set1 reg2, [reg1]” の形式
  第 2 オペランドのレジスタ値で指定したアドレスが示すデータの，第 1 オペランドで指定したレジスタ値で指定したビットをセットします。指定したビット以外は影響を受けません。

［詳細説明］

- disp に次のものを指定した場合，アセンブラでは，機械語命令の set1 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`set1 bit#3, disp16[reg1]`	`set1 bit#3, disp16[reg1]`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`set1 bit#3, $label[reg1]`	`set1 bit#3, $label[reg1]`

(c) !label，または %label を持つ相対値式

`set1 bit#3, !label[reg1]`	`set1 bit#3, !label[reg1]`
`set1 bit#3, %label[reg1]`	`set1 bit#3, %label[reg1]`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`set1 bit#3, HIGHW(value)[reg1]`	`set1 bit#3, HIGHW(value)[reg1]`
`set1 bit#3, LOWW(value)[reg1]`	`set1 bit#3, LOWW(value)[reg1]`
`set1 bit#3, HIGHW1(value)[reg1]`	`set1 bit#3, HIGHW1(value)[reg1]`

注 機械語命令の set1 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

`set1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `set1 bit#3, LOWW(disp)[r1]`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`set1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `set1 bit#3, LOWW(#label)[r1]`
`set1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `set1 bit#3, LOWW(label)[r1]`
`set1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `set1 bit#3, LOWW($label)[r1]`

- disp を省略した場合，アセンブラでは，0 が指定されたものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	指定したビットが 0 の場合 1，1 の場合 0 注
SAT	—

注 Z フラグの値は，この命令実行前の該当ビットの値を示しています。この命令実行後を示すものではありません。

clr1

ビット・クリアを行います。（Clear Bit）

［指定形式］

- clr1 bit#3, disp[reg1]
- clr1 reg2, [reg1]

disp に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

注意 “clr1 reg2, [reg1]” の形式では disp は指定できません。

［機能］

- “clr1 bit#3, disp[reg1]” の形式
  第 2 オペランドで指定したアドレスが示すデータの，第 1 オペランドに指定したビットをクリアします。指定したビット以外は影響を受けません。
- “clr1 reg2, [reg1]” の形式
  第 2 オペランドのレジスタ値で指定したアドレスが示すデータの，第 1 オペランドで指定したレジスタ値のビットをクリアします。指定したビット以外は影響を受けません。

［詳細説明］

- disp に次のものを指定した場合，アセンブラでは，機械語命令の clr1 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`clr1 bit#3, disp16[reg1]`	`clr1 bit#3, disp16[reg1]`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`clr1 bit#3, $label[reg1]`	`clr1 bit#3, $label[reg1]`

(c) !label，または %label を持つ相対値式

`clr1 bit#3, !label[reg1]`	`clr1 bit#3, !label[reg1]`
`clr1 bit#3, %label[reg1]`	`clr1 bit#3, %label[reg1]`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`clr1 bit#3, HIGHW(value)[reg1]`	`clr1 bit#3, HIGHW(value)[reg1]`
`clr1 bit#3, LOWW(value)[reg1]`	`clr1 bit#3, LOWW(value)[reg1]`
`clr1 bit#3, HIGHW1(value)[reg1]`	`clr1 bit#3, HIGHW1(value)[reg1]`

注 機械語命令の clr1 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

`clr1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `clr1 bit#3, LOWW(disp)[r1]`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`clr1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `clr1 bit#3, LOWW(#label)[r1]`
`clr1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `clr1 bit#3, LOWW(label)[r1]`
`clr1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `clr1 bit#3, LOWW($label)[r1]`

- disp を省略した場合，アセンブラでは，0 が指定されたものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分を省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	指定したビットが 0 の場合 1，1 の場合 0 注
SAT	—

注 Z フラグの値は，この命令実行前の該当ビットの値を示しています。この命令実行後を示すものではありません。

not1

ビット・ノットを行います。（Not Bit）

[指定形式]

- not1 bit#3, disp[reg1]
- not1 reg2, [reg1]

disp に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

注意 “not1 reg2, [reg1]” の形式では disp は指定できません。

[機能]

- “not1 bit#3, disp[reg1]” の形式
  第 2 オペランドで指定したアドレスが示すデータの，第 1 オペランドに指定したビットを反転（0 → 1，1 → 0）します。指定したビット以外は影響を受けません。
- “not1 reg2, [reg1]” の形式
  第 2 オペランドのレジスタ値で指定したアドレスが示すデータの，第 1 オペランドで指定したレジスタ値の示すビットを反転（0 → 1，1 → 0）します。指定したビット以外は影響を受けません。

[詳細説明]

- disp に次のものを指定した場合，アセンブラでは，機械語命令の not1 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`not1 bit#3, disp16[reg1]`	`not1 bit#3, disp16[reg1]`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`not1 bit#3, $label[reg1]`	`not1 bit#3, $label[reg1]`

(c) !label，または %label を持つ相対値式

`not1 bit#3, !label[reg1]`	`not1 bit#3, !label[reg1]`
`not1 bit#3, %label[reg1]`	`not1 bit#3, %label[reg1]`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`not1 bit#3, HIGHW(value)[reg1]`	`not1 bit#3, HIGHW(value)[reg1]`
`not1 bit#3, LOWW(value)[reg1]`	`not1 bit#3, LOWW(value)[reg1]`
`not1 bit#3, HIGHW1(value)[reg1]`	`not1 bit#3, HIGHW1(value)[reg1]`

注 機械語命令の not1 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

`not1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `not1 bit#3, LOWW(disp)[r1]`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`not1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `not1 bit#3, LOWW(#label)[r1]`
`not1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `not1 bit#3, LOWW(label)[r1]`
`not1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `not1 bit#3, LOWW($label)[r1]`

- disp を省略した場合，アセンブラでは，0 が指定されたものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	指定したビットが 0 の場合 1，1 の場合 0 注
SAT	—

注 Z フラグの値は，この命令実行前の該当ビットの値を示しています。この命令実行後を示すものではありません。

tst1

ビット・テストを行います。（Test Bit）

［指定形式］

- tst1 bit#3, disp[reg1]
- tst1 reg2, [reg1]

disp に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式
- 相対値式
- 上記のものに HIGHW，LOWW，または HIGHW1 を適用したもの

注意 “tst1 reg2, [reg1]” の形式では disp は指定できません。

［機能］

- “tst1 bit#3, disp[reg1]” の形式
  第 2 オペランドで指定したアドレスが示すデータの，第 1 オペランドに指定したビットの値に従い，フラグのみを設定します。第 2 オペランドの値，および指定したビットは変更されません。
- “tst1 reg2, [reg1]” の形式
  第 2 オペランドで指定したアドレスが示すデータの，第 1 オペランドで指定したレジスタ値が示すビットの値に従い，フラグのみを設定します。第 2 オペランドの値，および指定したビットは変更されません。

［詳細説明］

- disp に次のものを指定した場合，アセンブラでは，機械語命令の tst1 命令[注]が 1 つ生成されます。

(a) -32768 ～ +32767 の範囲の絶対値式

`tst1 bit#3, disp16[reg1]`	`tst1 bit#3, disp16[reg1]`

(b) sdata/sbss 属性セクションに定義を持つラベルの $label を持つ相対値式

`tst1 bit#3, $label[reg1]`	`tst1 bit#3, $label[reg1]`

(c) !label，または %label を持つ相対値式

`tst1 bit#3, !label[reg1]`	`tst1 bit#3, !label[reg1]`
`tst1 bit#3, %label[reg1]`	`tst1 bit#3, %label[reg1]`

(d) HIGHW，LOWW，または HIGHW1 を適用したもの

`tst1 bit#3, HIGHW(value)[reg1]`	`tst1 bit#3, HIGHW(value)[reg1]`
`tst1 bit#3, LOWW(value)[reg1]`	`tst1 bit#3, LOWW(value)[reg1]`
`tst1 bit#3, HIGHW1(value)[reg1]`	`tst1 bit#3, HIGHW1(value)[reg1]`

注 機械語命令の tst1 命令は，ディスプレースメントに -32768 ～ +32767（0xFFFF8000 ～ 0x7FFF）の範囲のイミーディエトをとります。

- disp に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) -32768 ～ +32767 の範囲を越える絶対値式

`tst1 bit#3, disp[reg1]`	`movhi HIGHW1(disp), reg1, r1` `tst1 bit#3, LOWW(disp)[r1]`

(b) #label，または label を持つ相対値式，および sdata/sbss 属性セクションに定義を持たないラベルの $label を持つ相対値式

`tst1 bit#3, #label[reg1]`	`movhi HIGHW1(#label), reg1, r1` `tst1 bit#3, LOWW(#label)[r1]`
`tst1 bit#3, label[reg1]`	`movhi HIGHW1(label), reg1, r1` `tst1 bit#3, LOWW(label)[r1]`
`tst1 bit#3, $label[reg1]`	`movhi HIGHW1($label), reg1, r1` `tst1 bit#3, LOWW($label)[r1]`

- disp を省略した場合，アセンブラでは，0 が指定されたものとみなされます。
- disp に #label を持つ相対値式，または #label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[r0] が指定されたものとみなされます。
- disp に $label を持つ相対値式，または $label を持つ相対値式に HIGHW，LOWW，または HIGHW1 を適用したものを指定した場合，その後ろの [reg1] の部分が省略できます。ただし，省略した場合，アセンブラでは，[gp] が指定されたものとみなされます。

［フラグ］

CY	—
OV	—
S	—
Z	指定したビットが 0 の場合 1，1 の場合 0
SAT	—

5.9.7 スタック操作命令

この項では，スタック操作命令について説明します。次に，この項において説明する命令を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.41 スタック操作命令

命令	意味
push	スタック領域へのプッシュ（単一レジスタ）
pushm	スタック領域へのプッシュ（複数レジスタ）
pop	スタック領域からのポップ（単一レジスタ）
popm	スタック領域からのポップ（複数レジスタ）

push

スタック領域へのプッシュを行います。(Push)

[指定形式]

push reg

[機能]

オペランドに指定したレジスタ値を，スタック領域にプッシュします。

[詳細説明]

- push 命令に対し，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

`push reg`	`add -4, sp` `st.w reg, [sp]`

[フラグ]

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

注意 命令展開が行われ，add 命令により設定されます。

pushm

スタック領域へのプッシュ（複数レジスタ）を行います。（Push Multiple）

[指定形式]

pushm reg1, reg2, …, regN

[機能]

オペランドに指定したレジスタ値を，スタック領域へプッシュします。なお，オペランドに指定可能なレジスタ数は，最大 32 個です。

[詳細説明]

- pushm 命令に対し，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。
  レジスタが 4 個以下の場合

pushm reg1, reg2, …, regN	add -4 * N, sp st.w regN, 4 * (N - 1)[sp] : st.w reg2, 4 * 1[sp] st.w reg1, 4 * 0[sp]

レジスタが 5 個以上の場合

pushm reg1, reg2, …, regN	addi -4 * N, sp, sp st.w regN, 4 * (N - 1)[sp] : st.w reg2, 4 * 1[sp] st.w reg1, 4 * 0[sp]

[フラグ]

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

注意 命令展開が行われ，add/addi 命令により設定されます。

pop

スタック領域からのポップを行います。（Pop）

[指定形式]

pop reg

[機能]

オペランドに指定したレジスタ値を，スタック領域からポップします。

[詳細説明]

- pop 命令に対し，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

`pop reg`	`ld.w [sp], reg` `add 4, sp`

[フラグ]

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

注意 命令展開が行われ，add 命令により設定されます。

popm

スタック領域からのポップ（複数レジスタ）を行います。（Pop Multiple）

[指定形式]

popm reg1, reg2, …, regN

[機能]

オペランドに指定したレジスタ値を，指定した順にスタック領域からポップします。なお，オペランドに指定可能なレジスタ数は，最大 32 個です。

[詳細説明]

- popm 命令に対し，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。
  レジスタが 3 個以下の場合

popm reg1, reg2, …, regN	ld.w 4 * 0[sp], reg1 ld.w 4 * 1[sp], reg2 : ld.w 4 * (N - 1)[sp], regN add 4 * N, sp

レジスタが 4 個以上の場合

popm reg1, reg2, …, regN	ld.w 4 * 0[sp], reg1 ld.w 4 * 1[sp], reg2 : ld.w 4 * (N - 1)[sp], regN addi 4 * N, sp, sp

[フラグ]

CY	MSB（Most Significant Bit）からのキャリーを生じた場合 1，そうでない場合 0
OV	Integer-Overflow を生じた場合 1，そうでない場合 0
S	結果が負になった場合 1，そうでない場合 0
Z	結果が 0 になった場合 1，そうでない場合 0
SAT	—

注意 命令展開が行われ，add/addi 命令により設定されます。

5.9.8 特殊命令

この項では，特殊命令について説明します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.42 特殊命令

命令	意味
ldsr	システム・レジスタへのロード
stsr	システム・レジスタの内容のストア
ldl.w	ダブルワード・データのロード
stc.w	ストア・コンディショナル
cll	アトミック操作のリンク解除
di	マスカブル割り込みの禁止
ei	マスカブル割り込みの許可
reti	トラップ，または割り込みルーチンからの復帰
eiret	EI レベル例外からの復帰
feret	FE レベル例外からの復帰
halt	プロセッサの停止
trap	ソフトウエア・トラップ
fetrap	FE レベル・ソフトウエア例外命令
nop	ノー・オペレーション
switch	テーブル参照分岐
callt	テーブル参照コール
ctret	callt からの復帰
caxi	比較と交換
rie	予約命令例外
syncm	メモリ同期化命令
syncp	パイプライン同期化命令
synce	例外同期化命令
synci	命令パイプライン同期化命令
prepare	スタック・フレームの生成（関数の前処理）
dispose	スタック・フレームの削除（関数の後処理）
syscall	システム・コール例外
pushsp	スタックへのプッシュ
popsp	スタックからのポップ
snooze	スヌーズ

ldsr

システム・レジスタへのロードを行います。（Load System Register）

［指定形式］

- ldsr reg, regID
- ldsr reg, regID, selID

regID，selID に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

［機能］

- “ldsr reg, regID” の形式
  第 1 オペランドに指定したレジスタ値を，第 2 オペランドに指定したシステム・レジスタ番号で示されるシステム・レジスタ注に格納します。
- “ldsr reg, regID, selID” の形式
  第 1 オペランドに指定したレジスタ値を，第 2 オペランドに指定したシステム・レジスタ番号と第 3 オペランドに指定したグループ番号で示されるシステム・レジスタ注に格納します。
  selID が省略された場合は，selID=0 が指定されたものとします。

注 システム・レジスタに関しては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

注意 システム・レジスタにプログラム・ステータス・ワード（PSW）を指定した場合，各フラグには reg に対応したビットを設定します。

［注意事項］

- regID，selID に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値を下位 5 ビット注を用いてアセンブルが続行されます。

  W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の ldsr 命令は，第 2 オペランドに 0 ～ 31（0x0 ～ 0x1F）の範囲のイミーディエトをとります。

- regID に予約レジスタ番号やアクセス禁止のレジスタ番号（ECR など）を指定した場合，またはデバッグ・モード時だけアクセス可能のレジスタ番号を指定した場合，次のメッセージが出力され，そのままアセンブルが続行されます。

  W0550018 : ldsr 命令に指定した番号のシステム・レジスタはアクセス禁止です。

stsr

システム・レジスタの内容のストアを行います。（Store System Register）

［指定形式］

- stsr regID, reg
- stsr regID, reg, selID

regID，selID に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

［機能］

- “stsr regID, reg” の形式
  第 1 オペランドに指定したシステム・レジスタ番号で示されるシステム・レジスタ注の値を，第 2 オペランドに指定したレジスタに格納します。
- “stsr regID, reg, selID” の形式
  第 1 オペランドに指定したシステム・レジスタ番号，第 3 オペランドに指定したグループ番号で示されるシステム・レジスタ注の値を，第 2 オペランドに指定したレジスタに格納します。
  selID が省略された場合は，selID=0 が指定されたものとします。

注 システム・レジスタに関しては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- regID，selID に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値の下位 5 ビット注を用いてアセンブルが続行されます。

  W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の stsr 命令は，第 1 オペランドに 0 ～ 31（0x0 ～ 0x1F）の範囲のイミーディエトをとります。

- regID に予約レジスタ番号を指定した場合，またはデバッグ・モード時だけアクセス可能のレジスタ番号を指定した場合，次のメッセージが出力され，そのままアセンブルが続行されます。

  W0550018 : ldsr 命令に指定した番号のシステム・レジスタはアクセス禁止です。

ldl.w

ダブルワード・データのロードを行います。（Load Linked）

［指定形式］

- ldl.w [reg1], reg2

［機能］

アトミック・リード・モディファイ・ライトのためにメモリからワード・データを読み出し，汎用レジスタ reg2 に格納します。さらに，指定したアドレスを含むアドレス範囲に対応するリンクを生成します。
以降，ldl.w 命令に対応する stc.w 命令が実行される以前に，特定の条件が成立した場合，リンクが消失します。リンクが消失した状態で，stc.w を実行した場合，stc.w の結果は，失敗を示します。
リンクが維持された状態で，stc.w を実行すると，stc.w の結果は成功となり，この場合も，リンクは消失します。
ldl.w 命令と stc.w 命令を使い，マルチコア・システムでのメモリ更新を正確に処理できます。

備考　マルチコア・システムでのアトミック性保証が必要なメモリ更新では，caxi 命令の代わりに ldl.w 命令と stc.w 命令を使用してください。

［詳細説明］

- アセンブラでは，機械語命令の ldl.w 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

stc.w

ストア・コンディショナルを行います。（Store Conditional）

[指定形式]

- stc.w reg2, [reg1]

[機能]

指定したアドレスに対応するリンクが存在する場合のみ成功となり，汎用レジスタ reg3 のワード・データをメモリに格納し，アトミック・リード・モディファイ・ライトを完了します。
すでにリンクが消失していた場合は，メモリへのストアは行わず，失敗となります。
stc.w 命令が成功したかどうかは，命令実行後の汎用レジスタ reg2 の内容で示されます。
stc.w 命令が成功すると，汎用レジスタ reg3 の内容はセット（1）され，stc.w 命令が失敗するとクリア（0）されます。
ldl.w 命令と stc.w 命令を使い，マルチコア・システムでのメモリ更新を正確に処理できます。

備考 マルチコア・システムでのアトミック性保証が必要なメモリ更新では，caxi 命令の代わりに ldl.w 命令と stc.w 命令を使用してください。

[詳細説明]

- アセンブラでは，機械語命令の stc.w 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

cll

アトミック操作のリンク解除を行います。（Clear Load Link）

［指定形式］

- cll

［機能］

ldl.w 命令で生成されるリンクを明示的に消失させます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—
ID	—

di

マスカブル割り込みの禁止を行います。（Disable Interrupt）

［指定形式］

- di

［機能］

PSW 中の ID ビットに 1 を設定し，本命令実行中からマスカブル割り込みの受け付けを禁止します。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—
ID	1

ei

マスカブル割り込みの許可を行います。（Enable Interrupt）

［指定形式］

- ei

［機能］

PSW 中の ID ビットに 0 を設定し，次の命令よりマスカブル割り込みの受け付けを許可します。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—
ID	0

reti

トラップ，または割り込みルーチンからの復帰を行います。（Return from Trap or Interrupt）

［指定形式］

- reti

［機能］

トラップ，または割り込みルーチンから復帰します。注

注 機能の詳細に関しては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	取り出した値
OV	取り出した値
S	取り出した値
Z	取り出した値
SAT	取り出した値

eiret

EI レベル例外からの復帰を行います。（Return from Trap or Interrupt）

［指定形式］

- eiret

［機能］

EI レベル例外から復帰する命令です。EIPC，EIPSW から復帰 PC と PSW を取り出し，PC，PSW に設定し制御を移します。
また，EP=0 の場合，割り込み（EIINT*n*）処理を終了したため，ISPR レジスタの所定のビットをクリアします。

［詳細説明］

- アセンブラでは，機械語命令の eiret 命令が 1 つ生成されます。

［フラグ］

CY	EIPSW から読み出した値が設定される
OV	EIPSW から読み出した値が設定される
S	EIPSW から読み出した値が設定される
Z	EIPSW から読み出した値が設定される
SAT	EIPSW から読み出した値が設定される

feret

FE レベル例外からの復帰を行います。（Return from Trap or Interrupt）

［指定形式］

- feret

［機能］

FE レベル例外から復帰する命令です。FEPC，FEPSW から復帰 PC と PSW を取り出し，PC，PSW に設定し制御を移します。

［詳細説明］

- アセンブラでは，機械語命令の feret 命令が 1 つ生成されます。

［フラグ］

CY	FEPSW から読み出した値が設定される
OV	FEPSW から読み出した値が設定される
S	FEPSW から読み出した値が設定される
Z	FEPSW から読み出した値が設定される
SAT	FEPSW から読み出した値が設定される

halt

停止します。（Halt）

［指定形式］

- halt

［機能］

プロセッサを停止し，HALT 状態に遷移します。なお，HALT 状態からの処理再開は，マスカブル割り込み，NMI，リセットにより行われます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

trap

ソフトウエア・トラップを発生させます。（Trap）

［指定形式］

- trap vector

vector に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

［機能］

ソフトウエア・トラップを発生させます。[注]

注 機能の詳細に関しては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- vector に 0 ～ 31 の範囲を越える絶対値式を指定した場合，次のメッセージが出力され，指定した値の下位 5 ビット[注]を用いてアセンブルが続行されます。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の trap 命令は，オペランドに 0 ～ 31（0x0 ～ 0x1F）の範囲のイミーディエトをとります。

fetrap

FE レベル・ソフトウエア例外命令を行います。（FE-level Trap）

[指定形式]

- fetrap

[機能]

復帰 PC（fetrap 命令の次の命令のアドレス）と現在の PSW の内容を，それぞれ FEPC と FEPSW に退避し，例外要因コードを FEIC レジスタに格納，PSW を例外要因にしたがって更新します。続いて，例外ハンドラ・アドレスに分岐し，例外処理を開始します。

vector4 と例外要因コード，例外ハンドラ・アドレスのオフセットの対応は表 5.43 vector4 と例外要因コード，例外ハンドラ・アドレスのオフセットの対応で示されます。例外ハンドラ・アドレスは，表 5.43 vector4 と例外要因コード，例外ハンドラ・アドレスのオフセットの対応が示すオフセット・アドレスを元に計算されます。

表 5.43 vector4 と例外要因コード，例外ハンドラ・アドレスのオフセットの対応

vector4	例外要因コード	オフセット・アドレス
0H	指定不可	
1H	00000031H	30H
2H	00000032H	
(中略)		
FH	0000003FH	

[詳細説明]

- アセンブラでは，機械語命令の fetrap 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

nop

ノー・オペレーションです。（No Operation）

［指定形式］

- nop

［機能］

何も行いません。命令シーケンス内に領域を確保したり，命令実行に遅延を挿入したりするために用いることができます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

switch

テーブル参照分岐を行います。（Jump With Table Look Up）

［指定形式］

switch reg

［機能］

次の順に処理を行います。

(1) オペランドで指定した値を 1 ビット論理左シフトした値とテーブルの先頭アドレス（switch 命令の次のアドレス）とを加算し，テーブル・エントリ・アドレスを生成します。

(2) 生成したテーブル・エントリ・アドレスから符号付きハーフワード・データをロードします。

(3) ロードした値を 1 ビット論理左シフトし，ワード長に符号拡張したあと，テーブルの先頭アドレスを加算し，アドレスを生成します。

(4) 生成したアドレスへ分岐します。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- reg に r0 を指定した場合，次のメッセージが出力され，アセンブルが中止されます。

E0550239 : RH850 指定時には、ソース・オペランドに r0 を指定することはできません。

callt

テーブル参照コールを行います。（Call With Table Look Up）

［指定形式］

- callt imm6

imm6 に指定できるものを次に示します。

- 6 ビット幅までの値を持つ絶対値式

［機能］

次の順に処理を行います。注

(1) 復帰 PC と PSW の値を CTPC と CTPSW に退避します。

(2) オペランドで指定された値を 1 ビット左シフトして CTBP（CALLT Base Pointer）からのオフセット値とし，CTBP 値と加算してテーブル・エントリ・アドレスを生成します。

(3) 生成したテーブル・エントリ・アドレスから符号なしハーフワード・データをロードします。

(4) ロードした値と CTBP 値を加算してアドレスを生成します。

(5) 生成したアドレスへ分岐します。

注 システム・レジスタについては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

ctret

callt から復帰します。（Return from Callt）

［指定形式］

- ctret

［機能］

callt による分岐から復帰します。次の順に処理を行います。注

(1) 復帰 PC と PSW を，CTPC と CTPSW から取り出します。

(2) 取り出した値を PC と PSW に設定し，制御を移します。

注 システム・レジスタについては，各デバイスのユーザーズ・マニュアルを参照してください。

［フラグ］

CY	取り出した値
OV	取り出した値
S	取り出した値
Z	取り出した値
SAT	取り出した値

caxi

比較と交換を行います。（Compare and Exchange for Interlock）

［指定形式］

- caxi [reg1], reg2, reg3

［機能］

指定したアドレスからワード・データを読み出し，汎用レジスタ reg2 のワード・データと比較し，結果を PSW の各フラグに示します。比較は汎用レジスタ reg2 のワード・データから，読み出したワード・データを減算することで行います。比較の結果が 0 であれば，汎用レジスタ reg3 のワード・データを，そうでなければ，読み出したワード・データを，生成したアドレスに格納します。

その後，読み出したワード・データを汎用レジスタ reg3 へ格納します。汎用レジスタ reg1，reg2 は影響を受けません。

［詳細説明］

- アセンブラでは，機械語命令の caxi 命令が 1 つ生成されます。

［フラグ］

CY	result の演算時に MSB へのボローがあれば 1，そうでないとき 0
OV	result の演算時にオーバーフローが起こった場合 1，そうでない場合 0
S	result が負になった場合 1，そうでない場合 0
Z	result が 0 になった場合 1，そうでない場合 0
SAT	—

rie

予約命令例外を行います。（Reserved Instruction Exception）

［指定形式］

- rie
- rie imm5, imm4

imm5 に指定できるものを次に示します。

- 5 ビット幅までの値を持つ絶対値式

imm4 に指定できるものを次に示します。

- 4 ビット幅までの値を持つ絶対値式

［機能］

復帰 PC（rie 命令のアドレス）と現在の PSW の内容を，それぞれ FEPC と FEPSW に退避し，例外要因コードを FEIC レジスタに格納，PSW を例外要因にしたがって更新します。続いて，例外ハンドラ・アドレスに分岐し，例外処理を開始します。
例外ハンドラ・アドレスは，オフセット・アドレス 60H を元に計算されます。

［詳細説明］

- アセンブラでは，機械語命令の rie 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

syncm

メモリ同期化命令を行います。（Synchronize Memory）

［指定形式］

- syncm

［機能］

CPU の実行パイプラインと，メモリの同期化を行います。
先行するすべてのメモリ・アクセスが " 同期化（synchronization）" されるまで実行の開始を待ち合わせます。
" 同期化 " とは，システム内のいずれのマスタ・デバイスから参照しても，syncm 命令に先行するメモリ・アクセスの結果が参照できる状態を指します。
バッファリング等によってメモリ・アクセスを遅延していて " 同期化 " されていない場合，syncm 命令は完了せず，" 同期化 " されるまで待ち合わせます。syncm 命令の実行が完了するまで，後続の命令は実行されません。
この命令は上記の機能がシステムとして提供されている場合に，マルチ・プロセッシング環境における「同期プリミティブ」の実現のための手段として使用が可能です。

［詳細説明］

- アセンブラでは，機械語命令の syncm 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

syncp

パイプライン同期化命令を行います。（Synchronize Pipline）

［指定形式］

- syncp

［機能］

この命令よりも前に実行されるべき，すべての命令の実行が完了するまで待ってから実行されます。

［詳細説明］

- アセンブラでは，機械語命令の syncp 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

sync e

例外同期化命令を行います。（Synchronize Exceptions）

[指定形式]

- sync e

[機能]

この命令以前の例外が同期化（synchronization）されるまで，実行の開始を待ち合わせます。
実行が開始された後，この命令は何も操作を行わず完了します。
“例外の同期化”とは，先行する命令に起因して発生する例外がすべて CPU に通知され，優先順位判定の対象となる時点まで待ち合わせることを指します。したがって，この命令の実行以前に例外の受け付け条件を満たしていた場合，先行する命令に起因して発生するすべてのインプレサイス例外（FPI 例外）は，必ずこの命令の実行完了前に受け付けられます。
この命令はマルチ・プロセッシング環境において，タスク切り替え／終了前の先行タスクの例外処理の完了を保証するために，利用することが可能です。

[詳細説明]

- アセンブラでは，機械語命令の sync e 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

synci

命令パイプライン同期化命令を行います。(Synchronize Instruction Pipeline)

[指定形式]

- synci

[機能]

先行するすべての命令の実行完了を待ち合わせます。synci 命令に後続する命令に対して synci 命令に先行する命令の効果が適応されることを保障します。この命令は，メモリ上で命令を書き換える「自己書き換えコード」の実現のための手段として使用が可能です。

備考 CPU の命令フェッチ・パイプラインをクリアし，後続命令の再フェッチを行います。
命令キャッシュを備えた CPU の場合，「自己書き換えコード」の実現には命令キャッシュの無効化も必要です。

[詳細説明]

- アセンブラでは，機械語命令の synci 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

prepare

スタック・フレームの生成（関数の前処理）を行います。（Function Prepare）

［指定形式］

- prepare list, imm1
- prepare list, imm1, imm2
- prepare list, imm1, sp

imm1/imm2 に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式

list は，prepare 命令でプッシュ可能な 12 本のレジスタを指定するものです。list に指定できるものを次に示します。

- レジスタ
  プッシュの対象となるレジスタ（r20 ～ r31）をカンマで区切って指定します。
- 12 ビット幅までの値を持つ絶対値式
  12 ビットと 12 本のレジスタとの対応は次のとおりです。
  ビット 11　　ビット 0

r30	r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31

次の 2 つの指定は等価です。

`prepare r26, r29, r31, 0x10`	`prepare 0x103, 0x10`

［機能］

prepare 命令は，関数の前処理をする命令です。

- “prepare list, imm1” の形式
  (a) 第 1 オペランドで指定したレジスタを 1 つプッシュし，スタック・ポインタ（sp）から 4 を減算します。
  (b) 第 1 オペランドで指定したレジスタをすべてプッシュし終わるまで（a）を繰り返します。
  (c) 第 2 オペランドで指定した絶対値式の値を sp から減算[注]し，sp をレジスタ退避領域に設定します。

- “prepare list, imm1, imm2” の形式
  (a) 第 1 オペランドで指定したレジスタを 1 つプッシュし，sp から 4 を減算します。
  (b) 第 1 オペランドで指定したレジスタをすべてプッシュし終わるまで（a）を繰り返します。
  (c) 第 2 オペランドで指定した絶対値式の値を sp から減算[注]し，sp をレジスタ退避領域に設定します。
  (d) 第 3 オペランドで指定した絶対値式の値を ep に設定します。

- “prepare list, imm1, sp” の形式
  (a) 第 1 オペランドで指定したレジスタを 1 つプッシュし，sp から 4 を減算します。
  (b) 第 1 オペランドで指定したレジスタをすべてプッシュし終わるまで（a）を繰り返します。
  (c) 第 2 オペランドで指定した絶対値式の値を sp から減算[注]し，sp をレジスタ退避領域に設定します。
  (d) 第 3 オペランドで指定した sp の値を ep に設定します。

注　機械語命令で実際に sp に減算される値は，imm1 を左へ 2 ビット・シフトした値となります。このためアセンブラは，指定した imm1 をあらかじめ右へ 2 ビット・シフトしてコードに反映します。

[詳細説明]

- imm1 に次のものを指定した場合，アセンブラでは，機械語命令の prepare 命令が 1 つ生成されます。

(a) 0 ～ 127 の範囲の絶対値式

prepare list, imm1	prepare list, imm1
prepare list, imm1, imm2	prepare list, imm1, imm2
prepare list, imm1, sp	prepare list, imm1, sp

- imm1 に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) 0 ～ 127 の範囲を越え，0 ～ 32767 の範囲の絶対値式

prepare list, imm1	prepare list, 0 movea -imm1, sp, sp
prepare list, imm1, imm2	prepare list, 0, imm2 movea -imm1, sp, sp
prepare list, imm1, sp	prepare list, 0, sp movea -imm1, sp, sp

(b) 0 ～ 32767 の範囲を越える絶対値式

prepare list, imm1	prepare list, 0
	mov imm1, r1
	sub r1, sp

prepare list, imm1, imm2	prepare list, 0, imm2
	mov imm1, r1
	sub r1, sp

prepare list, imm1, sp	prepare list, 0, sp
	mov imm1, r1
	sub r1, sp

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

dispose

スタック・フレームの削除（関数の後処理）を行います。（Function Dispose）

［指定形式］

- dispose imm, list
- dispose imm, list, [reg]

imm に指定できるものを次に示します。

- 32 ビット幅までの値を持つ絶対値式

list は，dispose 命令でポップ可能な 12 本のレジスタを指定するものです。list に指定できるものを，次に示します。

- レジスタ
  プッシュの対象となるレジスタ（r20 ～ r31）をカンマで区切って指定します。
- 12 ビット幅までの値を持つ絶対値式
  12 ビットと 12 本のレジスタとの対応は次のとおりです。

ビット 11　　ビット 0

r30	r24	r25	r26	r27	r20	r21	r22	r23	r28	r29	r31

次の 2 つの指定は等価です。

`dispose 0x10, r26, r29, r31`	`dispose 0x10, 0x103`

［機能］

dispose 命令は，関数の後処理をする命令です。

- “dispose imm, list” の形式

(a) 第 1 オペランドで指定した絶対値式の値をスタック・ポインタ（sp）に加算[注]し，sp をレジスタ退避領域に設定します。

(b) 第 2 オペランドで指定したレジスタを 1 つポップし，sp に 4 を加算します。

(c) 第 2 オペランドで指定したレジスタをすべてポップし終わるまで（b）を繰り返します。

- “dispose imm, list, [reg]” の形式

(a) 第 1 オペランドで指定した絶対値式の値をスタック・ポインタ（sp）に加算[注]し，sp をレジスタ退避領域に設定します。

(b) 第 2 オペランドで指定したレジスタを 1 つポップし，sp に 4 を加算します。

(c) 第 2 オペランドで指定したレジスタをすべてポップし終わるまで（b）を繰り返します。

(d) 第 3 オペランドで指定したレジスタ値をプログラム・カウンタ（PC）に設定します。

注　機械語命令で実際に sp に加算される値は，imm を左へ 2 ビット・シフトした値となります。したがって，アセンブラは，指定した imm をあらかじめ右へ 2 ビット・シフトしてコードに反映します。

[詳細説明]

- imm に次のものを指定した場合，アセンブラでは，機械語命令の dispose 命令が 1 つ生成されます。

(a) 0 ～ 127 の範囲の絶対値式

`dispose imm, list`	`dispose imm, list`
`dispose imm, list, [reg]`	`dispose imm, list, [reg]`

- imm に次のものを指定した場合，アセンブラでは，命令展開が行われ，複数個の機械語命令が生成されます。

(a) 0 ～ 127 の範囲を越え，0 ～ 32767 の範囲の絶対値式

`dispose imm, list`	`movea  imm, sp, sp` `dispose 0, list`
`dispose imm, list, [reg]`	`movea  imm, sp, sp` `dispose 0, list, [reg]`

(b) 0 ～ 32767 の範囲を越える絶対値式

`dispose imm, list`	`mov  imm, r1`
	`add  r1, sp`
	`dispose 0, list, [reg]`

`dispose imm, list, [reg]`	`mov  imm, r1`
	`add  r1, sp`
	`dispose 0, list, [reg]`

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- 形式 “dispose imm, list, [reg]” の [reg] に r0 を指定すると，次のメッセージが出力され，アセンブルが中止されます。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

syscall

システム・コール例外をを発生させます。（System Call）

[指定形式]

- syscall vector

vector に指定できるものを次に示します。

- 8 ビット幅までの値を持つ絶対値式

[機能]

OS のシステム・サービス呼び出しを行います。

(1) 復帰 PC（syscall 命令の次の命令のアドレス）と PSW の内容を EIPC と EIPSW に退避します。

(2) vector に対応する例外要因コードを，EIIC レジスタに格納します。
例外要因コードは，8000H に vector を加算した値です。

(3) PSW を例外要因にしたがって更新します。

(4) SCBP レジスタの値と，2 ビット論理左シフトしワード長までゼロ拡張した vector を加算して 32 ビット・テーブル・エントリ・アドレスを生成します。
ただし，vector がシステム・レジスタの SCCFG.SIZE ビットで指定された値より大きい場合，上記加算に用いる vector は 0 として扱います。

(5) (4) で生成されたアドレスのワードをロードします。

(6) (5) のデータに SCBP レジスタの値を加算した 32 ビット・ターゲット・アドレスを生成します。

(7) (6) で生成されたターゲット・アドレスへ分岐します。

[詳細説明]

- アセンブラでは，機械語命令の syscall 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- vector に 0 ～ 255 の範囲を越える絶対値式を指定した場合，次のメッセージを出力し，指定した値の下位 8 ビット[注]を用いてアセンブルを続行します。

W0550011 : イミーディエトの値が指定可能な値の範囲を越えています。

注 機械語命令の syscall 命令は，オペランドに 0 ～ 255（0x0 ～ 0xFF）範囲のイミーディエトをとります。

- OS のシステム・サービス呼び出しに使用する専用命令です。ユーザ・プログラム中での使用は，各 OS の機能仕様にしたがってください。
- syscall 命令のテーブル読み出しのためのメモリからの読み出し操作では，スーパバイザ特権でのメモリ保護が行われます。

pushsp

スタックへのプッシュを行います。（Push Registers from Stack）

［指定形式］

- pushsp rh, rt

rt，rh に指定できるものを次に示します。

- 汎用レジスタ r0 ～ r31

［機能］

汎用レジスタ rh から rt を昇順（rh，rh ＋ 1，rh ＋ 2，…，rt）にスタックに退避します。指定されたレジスタをすべて退避した後，sp を更新（減算）します。

備考　sp で指定された下位 2 ビットのアドレスは，0 にマスクされます。
また，sp の更新前に例外を受け付けると，実行を中止し，戻り番地をこの命令の先頭アドレスとして例外を処理してから，例外処理完了後に再実行します（sp は例外処理実行開始前の元の値を保持します）。

［詳細説明］

- アセンブラでは，機械語命令の pushsp 命令が 1 つ生成されます。

［フラグ］

CY	—
OV	—
S	—
Z	—
SAT	—

［注意事項］

- rh，rt に指定したレジスタ番号の大小関係が rh > rt となる場合は，次のメッセージが出力されます。

E0550249 : 構成に誤りがあります。

popsp

スタックからのポップを行います。(Pop Registers from Stack)

[指定形式]

- popsp rh, rt

rt，rh に指定できるものを次に示します。

- 汎用レジスタ r0 ～ r31

[機能]

汎用レジスタ rt から rh へ降順（rt，rt － 1，rt － 2，…，rh）にスタックからロードします。指定されたレジスタへすべてロードした後，sp を更新（加算）します。

備考　sp で指定された下位 2 ビットのアドレスは，0 にマスクされます。
また，sp の更新前に例外を受け付けると，実行を中止し，戻り番地をこの命令の先頭アドレスとして例外を処理してから，例外処理完了後に再実行します（sp は例外処理実行開始前の元の値を保持します）。

[詳細説明]

- アセンブラでは，機械語命令の popsp 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

[注意事項]

- rh，rt に指定したレジスタ番号の大小関係が rh > rt となる場合は，次のメッセージが出力されます。

```
E0550249 : 構成に誤りがあります。
```

- 復帰するレジスタに sp（r3）を含む指定をした場合（rh が 3 ～ 31），sp（r3）にはメモリから読み出してきた値は格納されません。このため，途中で中断した場合にも正しく再実行することが可能です。

snooze

スヌーズを行います。（Snooze）

[指定形式]

- snooze

[機能]

ハードウエア仕様によって定義された期間または特定の状態時に，snooze 命令を実行したスレッドの動作を一時停止させます。
定義された期間の経過または状態遷移時に自動的に復帰し，次命令から実行を開始します。
一時停止状態の解除条件は次のとおりです。

- 定義された一定期間の経過
- すべての中断型例外の発生

また，上記の例外の受け付け条件（ID および NP の値）を満たしていない場合であっても，要求が存在する場合には一時停止状態の解除が行われます（例：PSW.ID=1 であっても，INT0 が発生した段階で停止状態が解除されます）。
機能ごとに定義された次のマスク機能によって，中断型例外の発生がマスクされている場合は，一時停止状態は解除されません。

- 割り込みコントローラによる割り込みチャネルのマスク注
- 浮動小数点演算の例外許可ビットによるマスク
- 上記以外のハードウエア機能で定義されたマスク

注 ISPR レジスタ，PMR レジスタによるマスクは含まれません。

[詳細説明]

- アセンブラでは，機械語命令の snooze 命令が 1 つ生成されます。

[フラグ]

CY	—
OV	—
S	—
Z	—
SAT	—

5.9.9 ループ命令

次に，ループ命令の一覧を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.44 ループ命令

命令	意味
loop	ループ

loop

ループを行います。（Loop）

[指定形式]

- loop reg1, disp16

disp16 に指定できるものを次に示します。

- 16 ビット幅までの値を持つ絶対値式
- ラベルの PC オフセット参照を持つ相対値式
- label（PC オフセット）

[機能]

汎用レジスタ reg1 の内容から－1 を加算した値で汎用レジスタ reg1 を更新します。更新後の汎用レジスタ reg1 の内容が 0 でなかった場合，次の処理を行います。そうでないときは次の命令に進みます。

- 15 ビット・イミーディエトを 1 ビット論理左シフトしてワード長までゼロ拡張した結果を現在の PC から減算した値を PC に設定し，制御を移します。
- 汎用レジスタ reg1 の内容に対しては，－1（0xFFFFFFFF）を加算します。キャリー・フラグの更新は sub 命令ではなく，add 命令と同一の動作となります。

備考 16 ビット・ディスプレースメントのビット 0 は暗黙的に 0 が使用されます。なお，計算に使用される現在の PC とは，この命令自身の先頭バイトのアドレスであるためディスプレースメント値が 0 のときは，分岐先はこの命令自身になります。

[詳細説明]

- disp16 に次のものを指定した場合，本アセンブラは機械語命令の loop 命令が 1 つ生成されます。
  - (a) 0 ～ 65535 の範囲の絶対値式
  - (b) 本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち，0 ～ 65535 の範囲の相対値式

[フラグ]

CY	reg1 の演算時に MSB からのキャリーがあれば 1，そうでないとき 0
OV	reg1 の演算時にオーバーフローが起こった場合 1，そうでない場合 0
S	reg1 が負になった場合 1，そうでない場合 0
Z	reg1 が 0 になった場合 1，そうでない場合 0
SAT	—

[注意事項]

- disp16 に，0 ～ 65535 の範囲を越える絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち 0 ～ 65535 の範囲を越える相対値式を指定した場合，次のメッセージを出力しアセンブルを中止します。

E0550230 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- disp16 に，奇数値を持つ絶対値式，または本命令と同じファイル内の同じセクションに定義を持つラベルの PC オフセット参照を持ち奇数値を持つ相対値式を指定した場合，次のメッセージを出力し，アセンブルを中止します。

E0550226 : ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。

- loop 命令に対し，reg1 に r0 を指定した場合，次のメッセージを出力し，アセンブルを中止します。

E0550240 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

- reg1 には，r0 を指定しないでください。

5.9.10 浮動小数点演算命令

次に，浮動小数点演算命令の一覧を示します。
詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

表 5.45 浮動小数点演算命令（基本演算命令）

命令	意味
absf.d	浮動小数点絶対値（倍精度）
absf.s	浮動小数点絶対値（単精度）
addf.d	浮動小数点加算（倍精度）
addf.s	浮動小数点加算（単精度）
divf.d	浮動小数点除算（倍精度）
divf.s	浮動小数点除算（単精度）
maxf.d	浮動小数点最大値（倍精度）
maxf.s	浮動小数点最大値（単精度）
minf.d	浮動小数点最小値（倍精度）
minf.s	浮動小数点最小値（単精度）
mulf.d	浮動小数点乗算（倍精度）
mulf.s	浮動小数点乗算（単精度）
negf.d	浮動小数点符号反転（倍精度）
negf.s	浮動小数点符号反転（単精度）
recipf.d	逆数（倍精度）
recipf.s	逆数（単精度）
rsqrtf.d	平方根の逆数（倍精度）
rsqrtf.s	平方根の逆数（単精度）
sqrtf.d	平方根（倍精度）
sqrtf.s	平方根（単精度）
subf.d	浮動小数点減算（倍精度）
subf.s	浮動小数点減算（単精度）

表 5.46 浮動小数点演算命令（変換命令）

命令	意味
ceilf.dl	整数形式への変換（倍精度）
ceilf.dw	整数形式への変換（倍精度）
ceilf.dul	符号なし整数形式への変換（倍精度）
ceilf.duw	符号なし整数形式への変換（倍精度）
ceilf.sl	整数形式への変換（単精度）
ceilf.sw	整数形式への変換（単精度）
ceilf.sul	符号なし整数形式への変換（単精度）
ceilf.suw	符号なし整数形式への変換（単精度）
cvtf.dl	整数形式への変換（倍精度）
cvtf.ds	浮動小数点形式への変換（倍精度）
cvtf.dul	符号なし整数形式への変換（倍精度）
cvtf.duw	符号なし整数形式への変換（倍精度）
cvtf.dw	整数形式への変換（倍精度）
cvtf.ld	浮動小数点形式への変換（倍精度）
cvtf.ls	浮動小数点形式への変換（単精度）
cvtf.sd	浮動小数点形式への変換（倍精度）
cvtf.sl	整数形式への変換（単精度）
cvtf.sul	符号なし整数形式への変換（単精度）
cvtf.suw	符号なし整数形式への変換（単精度）
cvtf.sw	整数形式への変換（単精度）
cvtf.uld	浮動小数点形式への変換（倍精度）
cvtf.uls	浮動小数点形式への変換（単精度）
cvtf.uwd	浮動小数点形式への変換（倍精度）
cvtf.uws	浮動小数点形式への変換（単精度）
cvtf.wd	浮動小数点形式への変換（倍精度）
cvtf.ws	浮動小数点形式への変換（単精度）
floorf.dl	整数形式への変換（倍精度）
floorf.dw	整数形式への変換（倍精度）
floorf.dul	符号なし整数形式への変換（倍精度）
floorf.duw	符号なし整数形式への変換（倍精度）
floorf.sl	整数形式への変換（単精度）
floorf.sw	整数形式への変換（単精度）
floorf.sul	符号なし整数形式への変換（単精度）
floorf.suw	符号なし整数形式への変換（単精度）
trncf.dl	整数形式への変換（倍精度）

命令	意味
trncf.dul	符号なし整数形式への変換（倍精度）
trncf.duw	符号なし整数形式への変換（倍精度）
trncf.dw	整数形式への変換（倍精度）
trncf.sl	整数形式への変換（単精度）
trncf.sul	符号なし整数形式への変換（単精度）
trncf.suw	符号なし整数形式への変換（単精度）
trncf.sw	整数形式への変換（単精度）

表 5.47 浮動小数点演算命令（比較命令）

命令	意味
cmpf.d	浮動小数点比較（倍精度）
cmpf.s	浮動小数点比較（単精度）

表 5.48 浮動小数点演算命令（条件付き転送命令）

命令	意味
cmovf.d	条件付き転送（倍精度）
cmovf.s	条件付き転送（単精度）

表 5.49 浮動小数点演算命令（条件ビット転送命令）

命令	意味
trfsr	フラグ転送

absf.d

浮動小数点絶対値（倍精度）をとります。(Floating-point Absolute Value (Double))

[指定形式]

- absf.d reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容の絶対値をとり，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

[詳細説明]

- アセンブラでは，機械語命令の absf.d 命令が 1 つ生成されます。

absf.s

浮動小数点絶対値（単精度）をとります。（Floating-point Absolute Value (Single)）

[指定形式]

- absf.s reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容の絶対値をとり，汎用レジスタ reg2 に格納します。

[詳細説明]

- アセンブラでは，機械語命令の absf.s 命令が 1 つ生成されます。

addf.d

浮動小数点加算（倍精度）を行います。（Floating-point Add (Double)）

[指定形式]

- addf.d reg1, reg2, reg3

[機能]

汎用レジスタ reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容と汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を加算し，汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の addf.d 命令が 1 つ生成されます。

addf.s

浮動小数点加算（単精度）を行います。（Floating-point Add (Single)）

［指定形式］

- addf.s reg1, reg2, reg3

［機能］

汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を加算し，汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

［詳細説明］

- アセンブラでは，機械語命令の addf.s 命令が 1 つ生成されます。

divf.d

浮動小数点除算（倍精度）を行います。（Floating-point Divide (Double)）

[指定形式]

- divf.d reg1, reg2, reg3

[機能]

汎用レジスタ reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容で除算し，汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の divf.d 命令が 1 つ生成されます。

divf.s

浮動小数点除算（単精度）を行います。（Floating-point Divide (Single)）

［指定形式］

- divf.s reg1, reg2, reg3

［機能］

汎用レジスタ reg2 にある単精度浮動小数点形式の内容を汎用レジスタ reg1 にある単精度浮動小数点形式の内容で除算し，汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

［詳細説明］

- アセンブラでは，機械語命令の divf.s 命令が 1 つ生成されます。

maxf.d

浮動小数点最大値（倍精度）を求めます。(Floating-point Maximum (Double))

[指定形式]

- maxf.d reg1, reg2, reg3

[機能]

汎用レジスタ reg1 および reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式のデータの中から最大値を汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。
ソース・オペランドの 1 つが S-NaN の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，Q-NaN を格納します。

[詳細説明]

- アセンブラでは，機械語命令の maxf.d 命令が 1 つ生成されます。

maxf.s

浮動小数点最大値（単精度）を求めます。（Floating-point Maximum (Sngle)）

[指定形式]

- maxf.s reg1, reg2, reg3

[機能]

汎用レジスタ reg1 および reg2 にある単精度浮動小数点形式のデータの中から最大値を汎用レジスタ reg3 に格納します。
ソース・オペランドの 1 つが S-NaN の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，Q-NaN を格納します。

[詳細説明]

- アセンブラでは，機械語命令の maxf.s 命令が 1 つ生成されます。

minf.d

浮動小数点最小値（倍精度）を求めます。(Floating-point Minimum (Double))

[指定形式]

- minf.d reg1, reg2, reg3

[機能]

汎用レジスタ reg1 および reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式のデータの中から最小値を汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。
ソース・オペランドの 1 つが S-NaN の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，Q-NaN を格納します。

[詳細説明]

- アセンブラでは，機械語命令の minf.d 命令が 1 つ生成されます。

minf.s

浮動小数点最小値（単精度）を求めます。（Floating-point Minimum (Single)）

[指定形式]

- minf.s reg1, reg2, reg3

[機能]

汎用レジスタ reg1 および reg2 にある単精度浮動小数点形式のデータの中から最小値を汎用レジスタ reg3 に格納します。
ソース・オペランドの 1 つが S-NaN の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，Q-NaN を格納します。

[詳細説明]

- アセンブラでは，機械語命令の minf.s 命令が 1 つ生成されます。

mulf.d

浮動小数点乗算（倍精度）を行います。（Floating-point Multiply (Double)）

[指定形式]

- mulf.d reg1, reg2, reg3

[機能]

汎用レジスタ reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容と汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を乗算し，汎用レジスタ reg3 に格納します。

[詳細説明]

- アセンブラでは，機械語命令の mulf.d 命令が 1 つ生成されます。

mulf.s

浮動小数点乗算（単精度）を行います。（Floating-point Multiply (Single)）

［指定形式］

- mulf.s reg1, reg2, reg3

［機能］

汎用レジスタ reg2 にある単精度浮動小数点形式の内容と汎用レジスタ reg1 にある単精度浮動小数点形式の内容を乗算し，汎用レジスタ reg3 に格納します。

［詳細説明］

- アセンブラでは，機械語命令の mulf.s 命令が 1 つ生成されます。

negf.d

浮動小数点符号反転（倍精度）を行います。(Floating-point Negate (Double))

[指定形式]

- negf.d reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容の符号を反転し，汎用レジスタ reg2 に格納します。

[詳細説明]

- アセンブラでは，機械語命令の negf.d 命令が 1 つ生成されます。

negf.s

浮動小数点符号反転（単精度）を行います。(Floating-point Negate (Single))

[指定形式]

- negf.s reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容の符号を反転し，汎用レジスタ reg2 に格納します。

[詳細説明]

- アセンブラでは，機械語命令の negf.s 命令が 1 つ生成されます。

recipf.d

逆数（倍精度）を近似します。（Reciprocal of a Floating-point Value (Double)）

［指定形式］

- recipf.d reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容の逆数を近似し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。演算結果は divf 命令を用いた結果と異なります。

［詳細説明］

- アセンブラでは，機械語命令の recipf.d 命令が 1 つ生成されます。

recipf.s

逆数（単精度）を近似します。（Reciprocal of a Floating-point Value (Single)）

［指定形式］

- recipf.s reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容の逆数を近似し，汎用レジスタ reg2 に格納します。演算結果は divf 命令を用いた結果と異なります。

［詳細説明］

- アセンブラでは，機械語命令の recipf.s 命令が 1 つ生成されます。

rsqrtf.d

平方根の逆数（倍精度）を近似します。（Reciprocal of the Square Root of a Floating-point Value (Double)）

[指定形式]

- rsqrtf.d reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容の正の算術平方根の逆数を近似し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
演算結果は sqrtf，divf 命令を組み合わせた結果と異なります。

[詳細説明]

- アセンブラでは，機械語命令の rsqrtf.d 命令が 1 つ生成されます。

rsqrtf.s

平方根の逆数（単精度）を近似します。（Reciprocal of the Square Root of a Floating-point Value (Single)）

[指定形式]

- rsqrtf.s reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容の正の算術平方根の逆数を近似し，汎用レジスタ reg2 に格納します。
演算結果は sqrtf，divf 命令を組み合わせた結果と異なります。

[詳細説明]

- アセンブラでは，機械語命令の rsqrtf.s 命令が 1 つ生成されます。

sqrtf.d

平方根（倍精度）を求めます。(Floating-point Square Root (Double))

[指定形式]

- sqrtf.d reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容の正の算術平方根を求め，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。ソース・オペランドの値が－0 の場合，結果は－0 になります。

[詳細説明]

- アセンブラでは，機械語命令の sqrtf.d 命令が 1 つ生成されます。

sqrtf.s

平方根（単精度）を求めます。(Floating-point Square Root (Single))

[指定形式]

- sqrtf.s reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容の正の算術平方根を求め，汎用レジスタ reg2 に格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。ソース・オペランドの値が－0 の場合，結果は－0 になります。

[詳細説明]

- アセンブラでは，機械語命令の sqrtf.s 命令が 1 つ生成されます。

subf.d

浮動小数点減算（倍精度）を行います。（Floating-point Subtract (Double)）

［指定形式］

- subf.d reg1, reg2, reg3

［機能］

汎用レジスタ reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容から汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を減算し，汎用レジスタ reg3 で指定されるレジスタ・ペアに格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

［詳細説明］

- アセンブラでは，機械語命令の subf.d 命令が 1 つ生成されます。

subf.s

浮動小数点減算（単精度）を行います。(Floating-point Subtract (Single))

[指定形式]

- subf.s reg1, reg2, reg3

[機能]

汎用レジスタ reg2 にある単精度浮動小数点形式の内容から汎用レジスタ reg1 にある単精度浮動小数点形式の内容を減算し，汎用レジスタ reg3 に格納します。演算は無限精度であるかのように実行し，結果を現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の subf.s 命令が 1 つ生成されます。

ceilf.dl

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Long, round toward positive (Double)）

［指定形式］

- ceilf.dl reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の ceilf.dl 命令が 1 つ生成されます。

ceilf.dw

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Word, round toward positive (Double)）

［指定形式］

- ceilf.dw　reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が 2^{31} － 1 ～－ 2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞　：2^{31} － 1 を返します。
- ソースが負数，非数または－∞　：－ 2^{31} を返します。

［詳細説明］

- アセンブラでは，機械語命令の ceilf.dw 命令が 1 つ生成されます。

ceilf.dul

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Long, round toward positive (Double)）

［指定形式］

- ceilf.dul reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の ceilf.dul 命令が 1 つ生成されます。

ceilf.duw

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Word, round toward positive (Double)）

［指定形式］

- ceilf.duw reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の ceilf.duw 命令が 1 つ生成されます。

ceilf.sl

整数形式への変換（単精度）を行います。(Floating-point Convert Single to Long, round toward positive (Single))

[指定形式]

- ceilf.sl reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

[詳細説明]

- アセンブラでは，機械語命令の ceilf.sl 命令が 1 つ生成されます。

ceilf.sw

整数形式への変換（単精度）を行います。(Floating-point Convert Single to Word, round toward positive (Single))

[指定形式]

- ceilf.sw reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が 2^{31} － 1 ～－ 2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：2^{31} － 1 を返します。
- ソースが負数，非数または－∞ ：－ 2^{31} を返します。

[詳細説明]

- アセンブラでは，機械語命令の ceilf.sw 命令が 1 つ生成されます。

ceilf.sul

符号なし整数形式への変換（単精度）を行います。(Floating-point Convert Single to Unsigned-Long, round toward positive (Single))

[指定形式]

- ceilf.sul reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

[詳細説明]

- アセンブラでは，機械語命令の ceilf.sul 命令が 1 つ生成されます。

ceilf.suw

符号なし整数形式への変換（単精度）を行います。(Floating-point Convert Single to Unsigned-Word, round toward positive (Single))

[指定形式]

- ceilf.suw reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点数形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を＋∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

[詳細説明]

- アセンブラでは，機械語命令の ceilf.suw 命令が 1 つ生成されます。

cvtf.dl

整数形式への変換（倍精度）を行います。(Floating-point Convert Double to Long (Double))

[指定形式]

- cvtf.dl reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を現在の丸めモードに従って算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.dl 命令が 1 つ生成されます。

cvtf.ds

浮動小数点形式への変換（倍精度）を行います。（Floating-point Convert Double to Single (Double)）

[指定形式]

- cvtf.ds reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に単精度浮動小数点形式に変換し，汎用レジスタ reg2 に格納します。結果は，現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.ds 命令が 1 つ生成されます。

cvtf.dul

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Long (Double)）

[指定形式]

- cvtf.dul reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を現在の丸めモードに従って算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.dul 命令が 1 つ生成されます。

cvtf.duw

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Word (Double)）

［指定形式］

- cvtf.duw　reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。

ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞　：2^{32} － 1 を返します。
- ソースが負数，非数または－∞　：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.duw 命令が 1 つ生成されます。

cvtf.dw

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Word (Double)）

[指定形式]

- cvtf.dw reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。

ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.dw 命令が 1 つ生成されます。

cvtf.ld

浮動小数点形式への変換（倍精度）を行います。(Floating-point Convert Long to Double (Double))

[指定形式]

- cvtf.ld reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある 64 ビットの整数形式の内容を現在の丸めモードに従って算術的に倍精度浮動小数点形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.ld 命令が 1 つ生成されます。

cvtf.ls

浮動小数点形式への変換（単精度）を行います。(Floating-point Convert Long to Single (Single))

[指定形式]

- cvtf.ls reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある 64 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し，汎用レジスタ reg2 に格納します。結果は，現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.ls 命令が 1 つ生成されます。

cvtf.sd

浮動小数点形式への変換（倍精度）を行います。（Floating-point Convert Single to Double (Double)）

[指定形式]

- cvtf.sd reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を現在の丸めモードに従って算術的に倍精度浮動小数点形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.sd 命令が 1 つ生成されます。

cvtf.sl

整数形式への変換（単精度）を行います。（Floating-point Convert Single to Long (Single)）

［指定形式］

- cvtf.sl reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を現在の丸めモードに従って算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.sl 命令が 1 つ生成されます。

cvtf.sul

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Long (Single)）

［指定形式］

- cvtf.sul reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を現在の丸めモードに従って算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.sul 命令が 1 つ生成されます。

cvtf.suw

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Word (Single)）

［指定形式］

- cvtf.suw　reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。

ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞　：2^{32} － 1 を返します。
- ソースが負数，非数または－∞　：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.suw 命令が 1 つ生成されます。

cvtf.sw

整数形式への変換（単精度）を行います。(Floating-point Convert Single to Word (Single))

[指定形式]

- cvtf.sw reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。

ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。

無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.sw 命令が 1 つ生成されます。

cvtf.uld

浮動小数点形式への変換（倍精度）を行います。(Floating-point Convert Unsigned-Long to Double (Double))

[指定形式]

- cvtf.uld reg1, reg2

[機能]

汎用レジスタ reg1 で指定されるレジスタ・ペアにある符号のない 64 ビットの整数形式の内容を現在の丸めモードに従って算術的に倍精度浮動小数点形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.uld 命令が 1 つ生成されます。

cvtf.uls

浮動小数点形式への変換（単精度）を行います。（Floating-point Convert Unsigned-Long to Single (Single)）

［指定形式］

- cvtf.uls reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある符号のない 64 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し，汎用レジスタ reg2 に格納します。結果は，現在の丸めモードに従って丸めます。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.uls 命令が 1 つ生成されます。

cvtf.uwd

浮動小数点形式への変換（倍精度）を行います。（Floating-point Convert Unsigned-Word to Double (Double)）

［指定形式］

- cvtf.uwd　reg1, reg2

［機能］

汎用レジスタ reg1 にある符号のない 32 ビットの整数形式の内容を現在の丸めモードに従って算術的に倍精度浮動小数点形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
この変換演算は，精度落ちなく正確に実行されます。

［詳細説明］

- アセンブラでは，機械語命令の cvtf.uwd 命令が 1 つ生成されます。

cvtf.uws

浮動小数点形式への変換（単精度）を行います。（Floating-point Convert Unsigned-Word to Single (Single)）

[指定形式]

- cvtf.uws reg1, reg2

[機能]

汎用レジスタ reg1 にある符号のない 32 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し，汎用レジスタ reg2 に格納します。結果は，現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.uws 命令が 1 つ生成されます。

cvtf.wd

浮動小数点形式への変換（倍精度）を行います。（Floating-point Convert Word to Double (Double)）

[指定形式]

- cvtf.wd reg1, reg2

[機能]

汎用レジスタ reg1 にある 32 ビットの整数形式の内容を現在の丸めモードに従って算術的に倍精度浮動小数点形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
この変換演算は，精度落ちなく正確に実行されます。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.wd 命令が 1 つ生成されます。

cvtf.ws

浮動小数点形式への変換（単精度）を行います。(Floating-point Convert Word to Single (single))

[指定形式]

- cvtf.ws reg1, reg2

[機能]

汎用レジスタ reg1 にある 32 ビットの整数形式の内容を算術的に単精度浮動小数点形式に変換し，汎用レジスタ reg2 に格納します。結果は，現在の丸めモードに従って丸めます。

[詳細説明]

- アセンブラでは，機械語命令の cvtf.ws 命令が 1 つ生成されます。

floorf.dl

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Word, round toward negative (Double)）

［指定形式］

- floorf.dl reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.dl 命令が 1 つ生成されます。

floorf.dw

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Word, round toward negative (Double)）

［指定形式］

- floorf.dw reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.dw 命令が 1 つ生成されます。

floorf.dul

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Long, round toward negative (Double)）

［指定形式］

- floorf.dul reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.dul 命令が 1 つ生成されます。

floorf.duw

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Word, round toward negative (Double)）

［指定形式］

- floorf.duw reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.duw 命令が 1 つ生成されます。

floorf.sl

整数形式への変換（単精度）を行います。（Floating-point Convert Double to Long, round toward negative (Single)）

［指定形式］

- floorf.sl reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードにかかわりなく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.sl 命令が 1 つ生成されます。

floorf.sw

整数形式への変換（単精度）を行います。(Floating-point Convert Single to Word, round toward negative (Single))

[指定形式]

- floorf.sw reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

[詳細説明]

- アセンブラでは，機械語命令の floorf.sw 命令が 1 つ生成されます。

floorf.sul

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Long, round toward negative (Single)）

［指定形式］

- floorf.sul reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.sul 命令が 1 つ生成されます。

floorf.suw

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Word, round toward negative (Single)）

［指定形式］

- floorf.suw reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点数形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を－∞の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の floorf.suw 命令が 1 つ生成されます。

trncf.dl

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Long, round toward zero (Double)）

［指定形式］

- trncf.dl reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.dl 命令が 1 つ生成されます。

trncf.dul

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Long, round toward zero (Double)）

［指定形式］

- trncf.dul reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.dul 命令が 1 つ生成されます。

trncf.duw

符号なし整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Unsigned-Word, round toward zero (Double)）

［指定形式］

- trncf.duw reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.duw 命令が 1 つ生成されます。

trncf.dw

整数形式への変換（倍精度）を行います。（Floating-point Convert Double to Word, round toward zero (Double)）

［指定形式］

- trncf.dw reg1, reg2

［機能］

汎用レジスタ reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.dw 命令が 1 つ生成されます。

trncf.sl

整数形式への変換（単精度）を行います。（Floating-point Convert Single to Long, round toward zero (Single)）

［指定形式］

- trncf.sl　reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{63}-1$ ～ -2^{63} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞　：$2^{63}-1$ を返します。
- ソースが負数，非数または－∞　：-2^{63} を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.sl 命令が 1 つ生成されます。

trncf.sul

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Long, round toward zero (Single)）

［指定形式］

- trncf.sul reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点形式の内容を算術的に符号のない 64 ビットの整数形式に変換し，汎用レジスタ reg2 で指定されるレジスタ・ペアに格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{64} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{64} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{64} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.sul 命令が 1 つ生成されます。

trncf.suw

符号なし整数形式への変換（単精度）を行います。（Floating-point Convert Single to Unsigned-Word, round toward zero (Single)）

［指定形式］

- trncf.suw reg1, reg2

［機能］

汎用レジスタ reg1 にある単精度浮動小数点数形式の内容を算術的に符号のない 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数か負数の場合，または丸めた結果が 2^{32} － 1 ～ 0 の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが 2^{32} － 1 ～ 0 の範囲外の正数または＋∞ ：2^{32} － 1 を返します。
- ソースが負数，非数または－∞ ：0 を返します。

［詳細説明］

- アセンブラでは，機械語命令の trncf.suw 命令が 1 つ生成されます。

trncf.sw

整数形式への変換（単精度）を行います。(Floating-point Convert Single to Word, round toward zero (Single))

[指定形式]

- trncf.sw reg1, reg2

[機能]

汎用レジスタ reg1 にある単精度浮動小数点数形式の内容を算術的に 32 ビットの整数形式に変換し，汎用レジスタ reg2 に格納します。
現在の丸めモードに関係なく，結果を 0 の方向へ丸めます。
ソース・オペランドが無限大か非数の場合，または丸めた結果が $2^{31}-1$ ～ -2^{31} の範囲外の場合は，IEEE754 の無効演算例外を検出します。
無効演算例外の発生が許可されていない場合は，例外を発生せず，無効演算としてと FPSR レジスタの保存ビット（ビット 4）がセットされます。ソースの違いにより返す値は次のように異なります。

- ソースが正数または＋∞ ：$2^{31}-1$ を返します。
- ソースが負数，非数または－∞ ：-2^{31} を返します。

[詳細説明]

- アセンブラでは，機械語命令の trncf.sw 命令が 1 つ生成されます。

cmpf.d

浮動小数点比較（倍精度）を行います。（Floating-point Compare (Double)）

[指定形式]

- cmpf.d imm4, reg1, reg2, cc#3
- cmpf*cnd*.d reg1, reg2

imm4 に指定できるものを次に示します。

- 4 ビット幅までの値を持つ絶対値式

[機能]

- “cmpf.d imm4, reg1, reg2, cc#3” の形式
  reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を，比較条件 imm4 により，reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容と比較します。結果（真ならば 1，偽ならば 0）を cc#3 で指定される FPSR レジスタのコンディション・ビット（CC（7:0）ビット：ビット 31 ～ 24）にセットします。cc#3 が省略された場合は CC0 ビット（ビット 24）にセットします。
- “cmpf*cnd*.d reg1, reg2” の形式
  cmpf*cnd*.d により対応する cmpf.d 命令が生成され（「表 5.50 cmpfcnd.d 命令一覧」を参照），“cmpf.d imm4, reg1, reg2, cc#3” の形式に展開されます。reg2 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容を，比較条件により，reg1 で指定されるレジスタ・ペアにある倍精度浮動小数点形式の内容と比較します。結果（真ならば 1，偽ならば 0）を cc#3 で指定される FPSR レジスタのコンディション・ビット（CC（7:0）ビット：ビット 31 ～ 24）にセットします。cc#3 が省略された場合は CC0 ビット（ビット 24）にセットします。

[詳細説明]

- “cmpf.d imm4, reg1, reg2, cc#3” の形式の命令に対し，アセンブラでは，機械語命令の cmpf.d 命令が 1 つ生成されます。
- “cmpf*cnd*.d reg1, reg2” の形式の命令に対し，アセンブラでは，対応する cmpf.d 命令が生成され（「表 5.50 cmpfcnd.d 命令一覧」参照），“cmpf.d imm4, reg1, reg2, cc#3” の形式に展開されます。

表 5.50 cmpf*cnd*.d 命令一覧

命令	定義	説明	命令展開
cmpff.d	FALSE	常に偽	cmpf.d 0x0
cmpfun.d	Unordered	reg1, reg2 の少なくとも一方が非数	cmpf.d 0x1
cmpfeq.d	reg2 ＝ reg1	いずれも非数ではなく，かつ等しい	cmpf.d 0x2
cmpfueq.d	reg2 ? ＝ reg1	少なくとも一方が非数か，等しい	cmpf.d 0x3
cmpfolt.d	reg2 ＜ reg1	いずれも非数ではなく，かつより小さい	cmpf.d 0x4
cmpfult.d	reg2 ? ＜ reg1	少なくとも一方が非数か，より小さい	cmpf.d 0x5
cmpfole.d	reg2 ≦ reg1	いずれも非数ではなく，かつより小さいか，等しい	cmpf.d 0x6
cmpfule.d	reg2 ? ≦ reg1	少なくとも一方が非数か，より小さいか，等しい	cmpf.d 0x7
cmpfsf.d	FALSE	常に偽	cmpf.d 0x8
cmpfngle.d	Unordered	reg1, reg2 の少なくとも一方が非数	cmpf.d 0x9
cmpfseq.d	reg2 ＝ reg1	いずれも非数ではなく，かつ等しい	cmpf.d 0xA
cmpfngl.d	reg2 ? ＝ reg1	少なくとも一方が非数か，等しい	cmpf.d 0xB

命令	定義	説明	命令展開
cmpflt.d	reg2 ＜ reg1	いずれも非数ではなく，かつより小さい	cmpf.d 0xC
cmpfnge.d	reg2 ? ＜ reg1	少なくとも一方が非数か，より小さい	cmpf.d 0xD
cmpfle.d	reg2 ≦ reg1	いずれも非数ではなく，かつより小さいか，等しい	cmpf.d 0xE
cmpfngt.d	reg2 ? ≦ reg1	少なくとも一方が非数か，より小さいか，等しい	cmpf.d 0xF

備考 ?：Unordered（比較不能）

［注意事項］

- cmpf.d 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージを出力し，指定した値の下位 4 ビットを用いてアセンブルを続行します。

W0550011：イミーディエトの値が指定可能な値の範囲を越えています。

cmpf.s

浮動小数点比較（単精度）を行います。(Floating-point Compare (Single))

[指定形式]

- cmpf.s imm4, reg1, reg2, cc#3
- cmpf*cnd*.s reg1, reg2

imm4 に指定できるものを次に示します。

- 4 ビット幅までの値を持つ絶対値式

[機能]

- “cmpf.s imm4, reg1, reg2, cc#3” の形式
  reg2 で指定されるレジスタ・ペアにある単精度浮動小数点形式の内容を，比較条件 imm4 により，reg1 で指定されるレジスタ・ペアにある単精度浮動小数点形式の内容と比較します。結果（真ならば 1，偽ならば 0）を cc#3 で指定される FPSR レジスタのコンディション・ビット（CC（7:0）ビット：ビット 31 ～ 24）にセットします。cc#3 が省略された場合は CC0 ビット（ビット 24）にセットします。
- “cmpf*cnd*.s reg1, reg2” の形式
  cmpf*cnd*.s により対応する cmpf.s 命令が生成され（「表 5.51 cmpfcnd.s 命令一覧」を参照），“cmpf.s imm4, reg1, reg2, cc#3” の形式に展開されます。reg2 で指定されるレジスタ・ペアにある単精度浮動小数点形式の内容を，比較条件により，reg1 で指定されるレジスタ・ペアにある単精度浮動小数点形式の内容と比較します。結果（真ならば 1，偽ならば 0）を cc#3 で指定される FPSR レジスタのコンディション・ビット（CC（7:0）ビット：ビット 31 ～ 24）にセットします。cc#3 が省略された場合は CC0 ビット（ビット 24）にセットします。

[詳細説明]

- “cmpf.s imm4, reg1, reg2, cc#3” の形式の命令に対し，アセンブラでは，機械語命令の cmpf.s 命令が 1 つ生成されます。
- “cmpf*cnd*.s reg1, reg2” の形式の命令に対し，アセンブラでは，対応する cmpf.s 命令が生成され（「表 5.51 cmpfcnd.s 命令一覧」参照），“cmpf.s imm4, reg1, reg2, cc#3” の形式に展開されます。

表 5.51 cmpf*cnd*.s 命令一覧

命令	定義	説明	命令展開
cmpff.s	FALSE	常に偽	cmpf.s 0x0
cmpfun.s	Unordered	reg1, reg2 の少なくとも一方が非数	cmpf.s 0x1
cmpfeq.s	reg2 ＝ reg1	いずれも非数ではなく，かつ等しい	cmpf.s 0x2
cmpfueq.s	reg2 ? ＝ reg1	少なくとも一方が非数か，等しい	cmpf.s 0x3
cmpfolt.s	reg2 ＜ reg1	いずれも非数ではなく，かつより小さい	cmpf.s 0x4
cmpfult.s	reg2 ? ＜ reg1	少なくとも一方が非数か，より小さい	cmpf.s 0x5
cmpfole.s	reg2 ≦ reg1	いずれも非数ではなく，かつより小さいか，等しい	cmpf.s 0x6
cmpfule.s	reg2 ? ≦ reg1	少なくとも一方が非数か，より小さいか，等しい	cmpf.s 0x7
cmpfsf.s	FALSE	常に偽	cmpf.s 0x8
cmpfngle.s	Unordered	reg1, reg2 の少なくとも一方が非数	cmpf.s 0x9
cmpfseq.s	reg2 ＝ reg1	いずれも非数ではなく，かつ等しい	cmpf.s 0xA
cmpfngl.s	reg2 ? ＝ reg1	少なくとも一方が非数か，等しい	cmpf.s 0xB

命令	定義	説明	命令展開
cmpflt.s	reg2 ＜ reg1	いずれも非数ではなく，かつより小さい	cmpf.s 0xC
cmpfnge.s	reg2 ? ＜ reg1	少なくとも一方が非数か，より小さい	cmpf.s 0xD
cmpfle.s	reg2 ≦ reg1	いずれも非数ではなく，かつより小さいか，等しい	cmpf.s 0xE
cmpfngt.s	reg2 ? ≦ reg1	少なくとも一方が非数か，より小さいか，等しい	cmpf.s 0xF

備考 ?：Unordered（比較不能）

［注意事項］

- cmpf.s 命令の imm4 に 4 ビットの範囲を越える絶対値式を指定した場合，次のメッセージを出力し，指定した値の下位 4 ビットを用いてアセンブルを続行します。

W0550011：イミーディエトの値が指定可能な値の範囲を越えています。

cmovf.d

条件付き転送（倍精度）を行います。(Floating-point Conditional Move (Double))

[指定形式]

- cmovf.d cc#3, reg1, reg2, reg3

[機能]

オペコードの fcbit によって指定された FPSR.CC(7:0) ビットが真（1）の場合，reg1 で指定されるレジスタ・ペアのデータを reg3 で指定されるレジスタ・ペアに格納します。偽（0）の場合，reg2 で指定されるレジスタ・ペアのデータを reg3 で指定されるレジスタ・ペアに格納します。

[詳細説明]

- アセンブラでは，機械語命令の cmovf.d 命令が 1 つ生成されます。

[注意事項]

- cmovf.d 命令に対し，reg3 に r0 を指定した場合，次のメッセージを出力し，アセンブルを中止します。

E0550262 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

cmovf.s

条件付き転送（単精度）を行います。（Floating-point Conditional Move (Single)）

[指定形式]

- cmovf.s cc#3, reg1, reg2, reg3

[機能]

オペコードの fcbit によって指定された FPSR.CC(7:0) ビットが真（1）の場合，reg1 のデータを reg3 に格納します。偽（0）の場合，reg2 のデータを reg3 に格納します。

[詳細説明]

- アセンブラでは，機械語命令の cmovf.s 命令が 1 つ生成されます。

[注意事項]

- cmovf.s 命令に対し，reg3 に r0 を指定した場合，次のメッセージを出力し，アセンブルを中止します。

E0550262 : RH850 コア指定時には、デスティネーション・オペランドに r0 を指定することはできません。

trfsr

フラグ転送を行います。（Transfers specified CC bit to Zero flag in PSW (Single)）

［指定形式］

- trfsr cc#3
- trfsr

［機能］

fcbit によって指定された FPSR レジスタのコンディション・ビット（CC(7:0) ビット：ビット 31 ～ 24）を，PSW 内の Z フラグへ転送します。fcbit が省略された場合は CC0（ビット 24）を転送します。

［詳細説明］

- アセンブラでは，機械語命令の trfsr 命令が 1 つ生成されます。

5.9.11 その他の命令

次に，その他の命令を示します。

- 仮想化支援機能命令
- ハードウエア・マルチスレッディング機能命令
- キャッシュ命令
- MMU 制御命令
- SIMD 命令

詳細については，各デバイスのユーザーズ・マニュアル，およびアーキテクチャ編を参照してください。

6. セクション仕様

組み込み系のアプリケーションでは，プログラム・コードをある番地から配置したり，分割して配置するなど，メモリ配置に気を配る必要があります。
期待どおりのメモリ配置を実現するには，プログラム・コードやデータの配置情報を，最適化リンカに指示する必要があります。

6.1 セクション

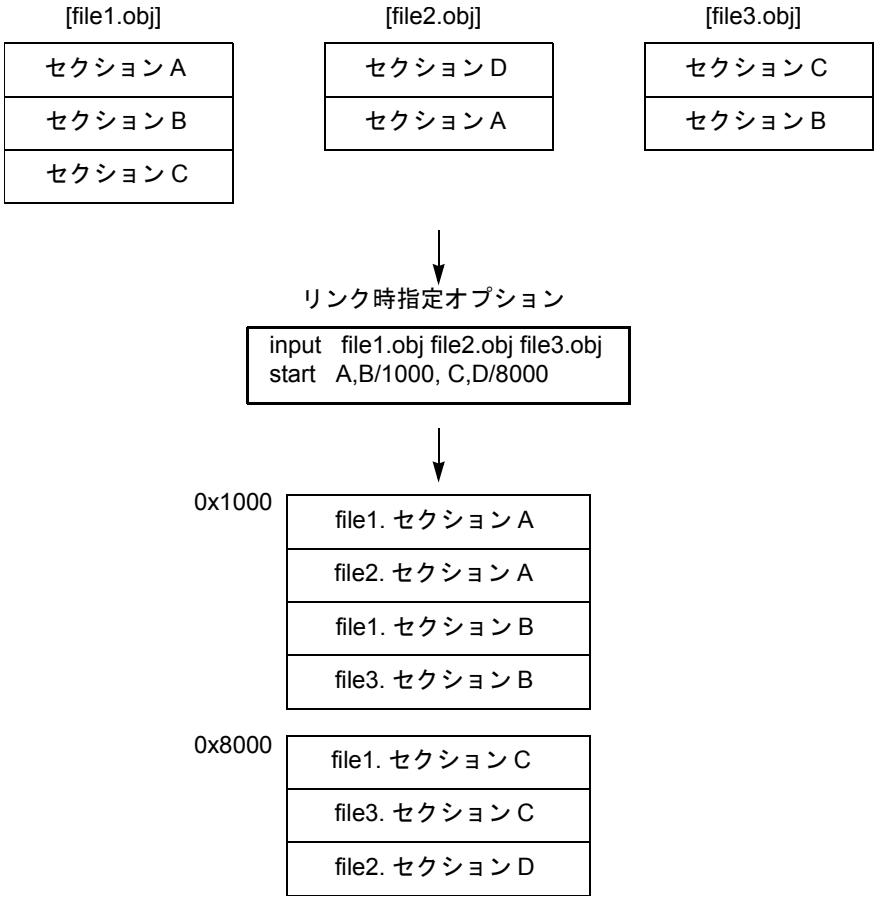
セクションとは，プログラムを構成する基本的な単位（プログラムやデータが配置される領域）です。たとえば，プログラム・コードは text 属性セクションへ，初期値を持つ変数は data 属性セクションへ，というように，セクションごとに分けて配置することになります。
セクション名はアプリケーション内で指定できます。C 言語では “#pragma section 指令 ”，アセンブリ言語では “ セクション定義疑似命令 ” によって指定できます。
ただし #pragma 指令でセクションの指定をしない場合でも，コンパイラはプログラム・コードやデータ（変数）にデフォルトとして決められたセクションを割り当てるようにしています。

6.1.1 セクションの結合

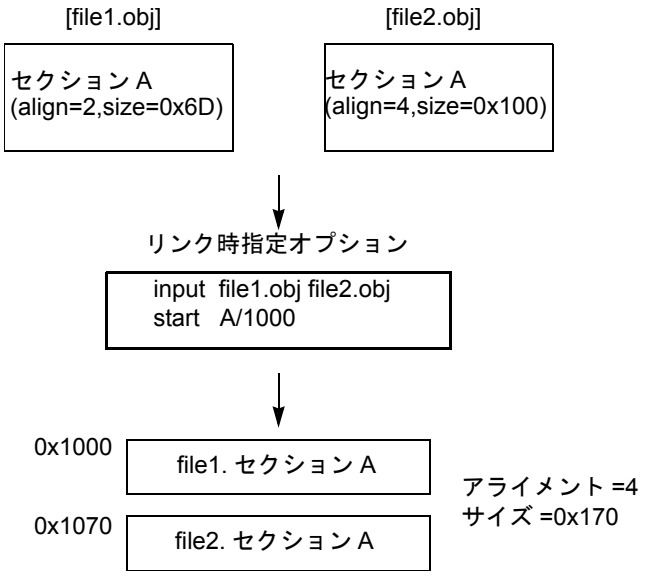
最適化リンカ（以降，“rlink” と略します）では，入力リロケータブルファイル内の同一セクションを結合し，-start オプションによって指定されたアドレスに割り付けます。

(1) -start オプションによるセクション配置

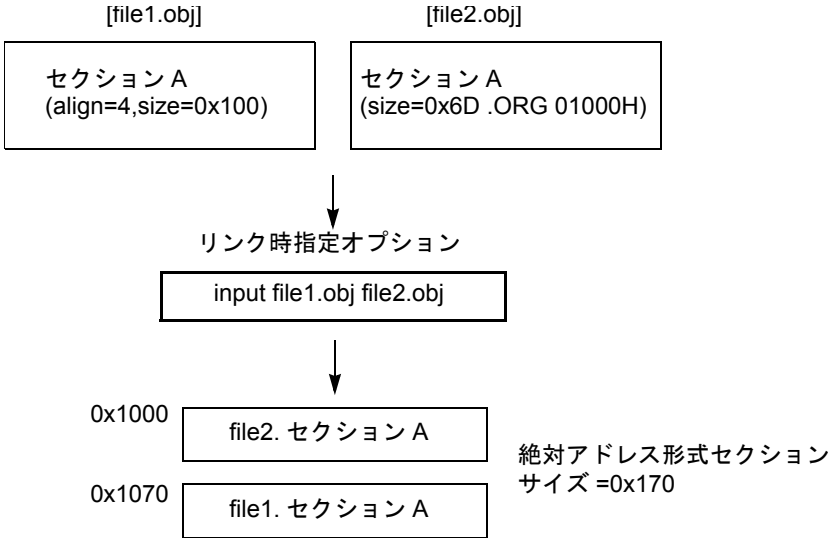
(a) 異なるファイルの同名セクションは，ファイルの入力順に連続して割り付けます。



(b) アライメント数の異なる同名セクションは，アライメント調整後に結合します。セクションのアライメント数は大きい方に合わせます。

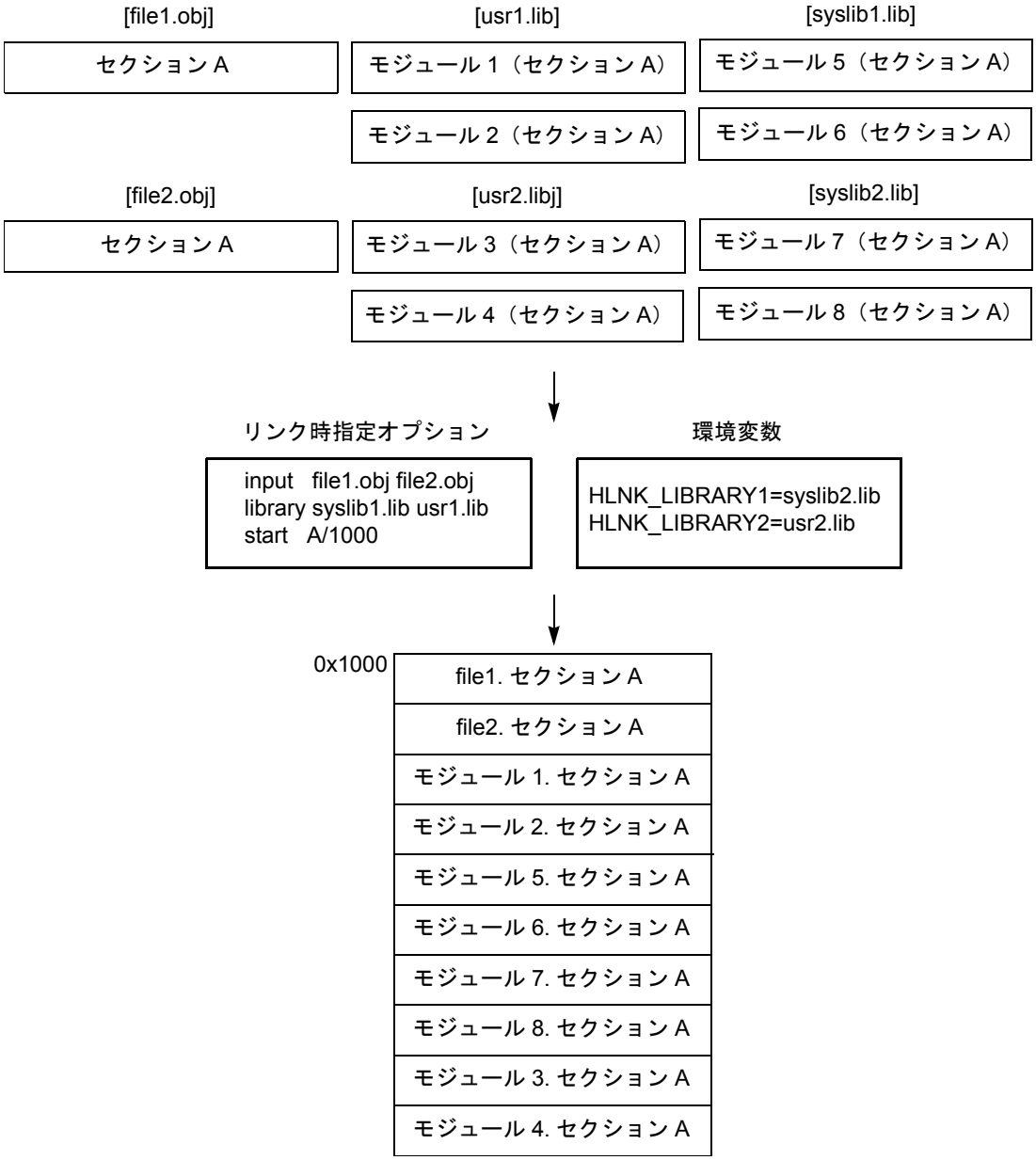


(c) 同名セクションに絶対アドレス形式と相対アドレス形式が含まれている場合，絶対アドレス形式セクションの後に相対アドレス形式セクションを結合します。



(d) 同名セクションの結合順序に関する規則は，優先度の高い順に以下のとおりです。

- input オプション，またはコマンド・ライン上の入力ファイル指定順
- library オプションのユーザ・ライブラリ指定順，およびライブラリ内モジュール入力順
- library オプションのシステム・ライブラリ指定順，およびライブラリ内モジュール入力順
- 環境変数 (HLNK_LIBRARY1 ～ 3) のライブラリ指定順，およびライブラリ内モジュール入力順



6.2 特殊シンボル

リンクの処理において，各出力セクションの先頭アドレス，各出力セクションの終端を越える最初のアドレス値を値として持つ予約シンボルを生成します。ユーザがこれらの予約シンボルと同名のシンボルを定義した場合，最適化リンカは定義されたシンボルを用い，独自に生成することはしません。

セクションの先頭アドレス値を値として持つ予約シンボルとしては，その出力セクションの名前の頭に “__s” を付けることによって構成される名前のシンボルが用いられます。

セクションの終端を越える最初のアドレス値を値として持つ予約シンボルとしては，その出力セクションの名前の頭に “__e” を付けることによって構成される名前のシンボルが用いられます。

以下に，予約されているセクションと，そのセクションに対する特殊シンボルを示します。

表 6.1 予約セクションに対する特殊シンボル

予約セクション	予約セクションに対する特殊シンボル
.text	__s.text, __e.text
.bss	__s.bss, __e.bss
.zbss	__s.zbss, __e.zbss
.zbss23	__s.zbss23, __e.zbss23
.ebss	__s.ebss, __e.ebss
.ebss23	__s.ebss23, __e.ebss23
.tbss4	__s.tbss4, __e.tbss4
.tbss5	__s.tbss5, __e.tbss5
.tbss7	__s.tbss7, __e.tbss7
.tbss8	__s.tbss8, __e.tbss8
.sbss	__s.sbss, __e.sbss
.sbss23	__s.sbss23, __e.sbss23
.data	__s.data, __e.data
.zdata	__s.zdata, __e.zdata
.zdata23	__s.zdata23, __e.zdata23
.edata	__s.edata, __e.edata
.edata23	__s.edata23, __e.edata23
.tdata	__s.tdata, __e.tdata
.tdata4	__s.tdata4, __e.tdata4
.tdata5	__s.tdata5, __e.tdata5
.tdata7	__s.tdata7, __e.tdata7
.tdata8	__s.tdata8, __e.tdata8
.sdata	__s.sdata, __e.sdata
.sdata23	__s.sdata23, __e.sdata23
.const	__s.const, __e.const
.zconst	__s.zconst, __e.zconst
.zconst23	__s.zconst23, __e.zconst23

注意 生成するシンボルは，表のうち，リンク処理後の実行形式ファイルにセクションが存在するもののみとなります。

7. ライブラリ関数仕様

この章では，CC-RH が提供するライブラリ関数について説明します。

7.1 提供ライブラリ

CC-RH で提供しているライブラリは，次のとおりです。

表 7.1 提供ライブラリ

提供ライブラリ	ライブラリ名	概要
標準ライブラリ	libc.lib	プログラム診断機能関数 可変個引数関数 文字列関数 メモリ管理関数 文字変換関数 文字分類関数 標準入出力関数 標準ユーティリティ関数 周辺装置の初期化関数 RAM セクション領域初期化関数 演算用ランタイム関数
	libsetjmp.lib	非局所分岐関数
数学ライブラリ（倍精度）	libm.lib	数学関数
数学ライブラリ（単精度）	libmf.lib	
標準ライブラリ，数学ライブラリ（倍精度），数学ライブラリ（単精度）	rhf8n.lib rhs8n.lib rhf4n.lib rhs4n.lib rhf8z.lib rhf4z.lib	プログラム診断機能関数 可変個引数関数 文字列関数 メモリ管理関数 文字変換関数 文字分類関数 標準入出力関数 標準ユーティリティ関数 周辺装置の初期化関数 RAM セクション領域初期化関数 演算用ランタイム関数 数学関数

標準ライブラリや数学ライブラリを，アプリケーション内で使用するときは，関連するヘッダ・ファイルをインクルードして，ライブラリ関数を使用します。

ただし，プログラム診断機能関数，可変個引数関数，文字変換関数，文字分類関数だけを使用している場合，ライブラリの参照を行う必要はありません。

CS+ を使用する場合，これらのライブラリはデフォルトで参照する設定になっています。

演算用ランタイム関数は，浮動小数点演算や整数演算を行うときに，CC-RH が自動的に呼び出すルーチンです。

したがって，“演算用ランタイム関数”は，他のライブラリ関数とは異なり，C ソースやアセンブラ・ソースで記述する関数ではありません。

rh***.lib は libc.lib, libm.lib, libmf.lib の内容を含む（libsetjmp.lib は含まない）ので，rh***.lib 指定時は libc.lib，libm.lib, libmf.lib を同時に指定しないでください。また rh***.lib は 1 つだけ指定してください。それ以外の指定をした場合の動作は保証しません。

7.2 ヘッダ・ファイル

CC-RH でライブラリを使用するときに必要となるヘッダ・ファイルの一覧は次のとおりです。

なお，各ファイルにはマクロ定義，関数宣言が記述されています。

表 7.2 ヘッダ・ファイル

ファイル名	概要
assert.h	プログラム診断機能のためのヘッダ・ファイル
ctype.h	文字の変換，分類のためのヘッダ・ファイル
errno.h	エラー条件の報告のためのヘッダ・ファイル
float.h	浮動小数点表現，浮動小数点演算のためのヘッダ・ファイル
limits.h	整数の数量的限界のためのヘッダ・ファイル
math.h	数学計算のためのヘッダ・ファイル
mathf.h	数学計算のためのヘッダ・ファイル（ANSI 規格規定外の単精度形式の数学関数の宣言とマクロの定義 ）
setjmp.h	非局所分岐のためのヘッダ・ファイル
stdarg.h	可変個の引数を持つ関数をサポートするためのヘッダ・ファイル
stddef.h	共通の定義のためのヘッダ・ファイル
stdio.h	標準入出力のためのヘッダ・ファイル
stdlib.h	標準ユーティリティのためのヘッダ・ファイル
string.h	メモリ操作，文字列操作のためのヘッダ・ファイル
_h_c_lib.h	初期設定用ルーチンのためのヘッダ・ファイル

7.3 リエントラント性

“ リエントラント ” とは “ 再入可能 ” という意味です。リエントラント性を持つ関数は，その関数実行中に，他のプロセスでもその関数実行しようとした場合でも，正しく実行できます。たとえば，リアルタイム OS を用いたアプリケーションで，あるタスクがこの関数を実行している最中に，割り込みなどがトリガになって他のタスクへディスパッチし，そこでもこの関数を実行しても正しく実行されます。関数が RAM をテンポラリとして使う必要のある関数は，リエントラント性を持たないことがあります。

7.4 ライブラリ関数

この節では，ライブラリ関数について説明します。

7.4.1 プログラム診断機能関数

プログラム診断機能関数として，以下のものがあります。

表 7.3 プログラム診断機能関数

関数／マクロ名	概要
assert	プログラム中に診断機能を付加

assert

プログラム中に診断機能を付け加えます。

[所属]

標準ライブラリ

[指定形式]

#include <asssert.h>
assert(int *expression*);

[詳細説明]

expression が真の時は値を返さずに処理を終了します。expression が偽の時は，診断情報をコンパイラによって定義された書式で標準エラー・ファイルに出力し，その後 abort 関数[注]を呼び出します。
診断情報の中には，パラメータのプログラム・テキスト，ソース・ファイル名，ソース行番号が含まれています。

assert マクロを無効にする場合，assert.h を取り込む前に #define NDEBUG を定義してください。

注 assert マクロを使用する場合，ユーザ・システムに合わせて abort 関数を作成する必要があります。

[使用例]

```
#include    <assert.h>
int func(void);
int main() {
    int ret;
    ret = func();
    assert(ret == 0);        ← ret が 0 以外のとき，abort() が呼ばれる
    return 0;
}
```

7.4.2 可変個引数関数

可変個引数関数として，以下のものがあります。

表 7.4 可変個引数関数

関数／マクロ名	概要
va_start	引数リスト走査用変数の初期化
va_end	引数リスト走査の終了
va_arg	引数リスト走査用変数の移動

va_start

引数リスト走査用変数の初期化を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdarg.h>
void va_start(va_list *ap*, *last-named-argument*);

[詳細説明]

変数 *ap* を，可変個引数リストの先頭（*last-named-argument* の次の引数）を指すように初期化します。
なお，可変個の引数を持つ関数 *func* を，移植性を持つ形で定義するには，次に示した形式を用います。

```
#include    <stdarg.h>
void func(arg-declarations, ...) {
    va_list ap;
    type    argN;
    va_start(ap, last-named-argument);
    argN = va_arg(ap, type);
    va_end(ap);
}
```

備考 *arg-declarations* は，引数リストで，最後に *last-named-argument* が宣言されているものとします。後ろに続く “, ...” は可変個引数リストを示します。va_list は，引数リストの走査に用いられる変数（この例の場合 *ap*）の型です。

[使用例]

```
#include    <stdarg.h>
void abc(int first, int second, ...) {
    va_list ap;
    int     i;
    char    c, *fmt;
    va_start(ap, second);
    i = va_arg(ap, int);
    c = va_arg(ap, int);    /*char 型は int 型に変換 */
    fmt = va_arg(ap, char *);
    va_end(ap);
}
```

va_end

引数リスト走査の終了を示します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdarg.h>
void    va_end(va_list ap);
```

[詳細説明]

リストの走査の終了を示します。va_arg を va_start と本関数とで囲むことにより，リストの走査を繰り返すことができます。

va_arg

引数リスト走査用変数の移動を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdarg.h>
type    va_arg(va_list ap, type);
```

[詳細説明]

変数 *ap* の指している引数を返し，次の引数を指すように変数 *ap* を進めます。*type* には，引数が関数に渡される際に変換される型を指定します。コンパイラでは，singed char 型，および short 型の引数に対しては int 型を指定し，unsinged char 型，および unsigned short 型の引数に対しては unsigned int 型を指定してください。引数ごとに異なる型を指定することができますが，“どの型の引数が渡されてきているか”は，呼び出された側と呼び出し側の関数との間の取り決めによって規定されるようにしてください。

また，“実際に引数がいくつ渡されてきているか”に関しても，呼び出された側と呼び出し側の関数との間の取り決めによって規定されるようにしてください。

7.4.3 文字列関数

文字列関数として，以下のものがあります。

表 7.5 文字列関数

関数／マクロ名	概要
strpbrk	文字列検索（最初の位置）
strrchr	文字列検索（最後の位置）
strchr	文字列検索（指定文字の最初の位置）
strstr	文字列検索（指定文字列の最初の位置）
strspn	文字列検索（指定文字を含む最大の長さ）
strcspn	文字列検索（指定文字を含まない最大の長さ）
strcmp	文字列比較
strncmp	文字列比較（文字数指定）
strcpy	文字列コピー
strncpy	文字列コピー（文字数指定）
strcat	文字列連結
strncat	文字列連結（文字数指定）
strtok	トークン分割
strlen	文字列の長さ
strerror	エラー番号の文字列変換

strpbrk

文字列検索（最初の位置）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strpbrk(const char *s1, const char *s2);
```

[戻り値]

この文字を指すポインタを返します。*s2* からの文字がいずれも *s1* の中に現れなかつた場合は，null ポインタを返します。

[詳細説明]

s2 の指す文字列中のいずれかの文字（null 文字（¥0）は除く）が *s1* の指す文字列中に最初に現れた位置を求めます。

strrchr

文字列検索（最後の位置）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strrchr(const char *s, int c);
```

[戻り値]

見つかった *c* を指すポインタを返します。*c* がこの文字列中に現れなかった場合，null ポインタを返します。

[詳細説明]

char 型に変換された *c* が *s* の指す文字列中に最後に現れた位置を求めます。終端を示す null 文字（¥0）は，この文字列の一部であるとみなされます。

strchr

文字列検索（指定文字の最初の位置）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strchr(const char *s, int c);
```

[戻り値]

見つかった文字を指すポインタを返します。*c* がこの文字列中に現れなかった場合は，null ポインタを返します。

[詳細説明]

char 型に変換された *c* と同じ文字が，*s* の指す文字列中に最初に現れる位置を求めます。終端を示す null 文字（¥0）は，この文字列の一部であるとみなされます。

strstr

文字列検索（指定文字列の最初の位置）を行います。

［所属］

標準ライブラリ

［指定形式］

```
#include <string.h>
char    *strstr(const char *s1, const char *s2);
```

［戻り値］

見つかった文字列を指すポインタを返します。文字列 *s2* が見つからなかった場合，null ポインタを返します。*s2* が長さゼロの文字列を指している場合，*s1* を返します。

［詳細説明］

s1 の指す文字列の中で，*s2* の指す文字列と最初に一致する部分（null 文字（¥0）は除く）の位置を求めます。

strspn

文字列検索（指定文字を含む最大の長さ）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
size_t  strspn(const char *s1, const char *s2);
```

[戻り値]

先頭部分の長さを返します。

[詳細説明]

s1 の指す文字列の中で，*s2* の指す文字列中にある文字（null 文字（¥0）は除く）のみで構成されている先頭部分の長さを求めます。

strcspn

文字列検索（指定文字を含まない最大の長さ）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
size_t  strcspn(const char *s1, const char *s2);
```

[戻り値]

先頭部分の長さを返します。

[詳細説明]

s1 の指す文字列の中で，*s2* の指す文字列（終わりの null 文字（¥0）は除く）の中にない文字のみで構成されている，先頭部分の長さを求めます。

strcmp

文字列比較を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

[戻り値]

s1 の指す文字列が *s2* の指す文字列と比べて大きい，等しい，または小さいかによって，0 より大きい，0 に等しい，0 より小さい整数を返します。

[詳細説明]

s1 の指す文字列と *s2* の指す文字列とを比較します。

strncmp

文字列比較（文字数指定）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t length);
```

[戻り値]

s1 の指す配列が *s2* の指す配列より大きい，等しい，または小さいかによって，0 より大きい，0 に等しい，0 より小さい整数を返します。

[詳細説明]

s1 の指す配列の文字と *s2* の指す配列の文字を最大で *length* 文字比較します。

strcpy

文字列コピーを行います。

[所属]

標準ライブラリ

[指定形式]

#include <string.h>
char *strcpy(char **dst*, const char **src*);

[戻り値]

dst の値を返します。

[詳細説明]

src の指す文字列を *dst* の指す配列にコピーします。

[使用例]

```
#include    <string.h>
void func(char *str, const char *src) {
    strcpy(str, src);   /*src の指す文字列を str の指す配列にコピー */
      :
}
```

strncpy

文字列コピー（文字数指定）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strncpy(char *dst, const char *src, size_t length);
```

[戻り値]

dst の値を返します。

[詳細説明]

src の指す配列から *dst* の指す配列に最大で *length* 文字（null 文字（¥0）を含む）コピーします。*src* の指す配列が *length* 文字より短い文字列の場合，全部で *length* 文字分書き込まれるまで，*dst* の指す配列内のコピーに null 文字（¥0）が付加されます。

strcat

文字列連結を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strcat(char *dst, const char *src);
```

[戻り値]

dst の値を返します。

[詳細説明]

src の指す文字列のコピーを，null 文字（¥0）を含めて，*dst* の指す文字列の末尾に連結します。*src* の最初の文字は *dst* の終わりの null 文字（¥0）を上書きします。

strncat

文字列連結（文字数指定）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strncat(char *dst, const char *src, size_t length);
```

[戻り値]

dst の値を返します。

[詳細説明]

src の指す文字列の先頭から，最大で *length* 文字（*src* の null 文字（¥0）を含む）を *dst* の指す文字列の末尾に連結します。*src* の最初の文字は *dst* の終わりの null 文字（¥0）を上書きします。この結果には，終端を示す null 文字（¥0）が常に付加されます。

[注意事項]

null 文字（¥0）は常に付加されるので，コピーが *length* によって制限される場合，*dst* に付加される文字の個数は *length* + 1 になることに注意してください。

strtok

トークン分割を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strtok(char *s, const char *delimiters);
```

[戻り値]

トークンへのポインタを返します。トークンが存在しない場合は，null ポインタを返します。

[詳細説明]

s の指す文字列を，*delimiters* の指す文字列中の文字で区切ることによって，トークンの列に分割します。これは最初に呼び出されると，最初の引数として *s* を持ち，その後は null ポインタを最初の引数とする呼び出しが続きます。*delimiters* の指す区切り文字列は，呼び出しごとに異なっていてもかまいません。最初の呼び出しでは，*delimiters* の指す区切り文字列中に含まれない最初の文字を求めて *s* の指す文字列中をサーチします。そのような文字が見つからなかった場合，null ポインタを返します。そのような文字が見つかった場合，その文字が最初のトークンの始まりとなります。その後，そのときの区切り文字列に含まれる文字を求めてそこからサーチを行います。

そのような文字が見つからなかった場合，そのときのトークンは *s* の指す文字列の終わりまで拡張され，あとに続くサーチは null ポインタを返します。そのような文字が見つかった場合，その文字はトークンの終端を示す null 文字（¥0）で上書きされます。本関数は，あとに続く文字を指すポインタをセーブします。null ポインタを最初の引数の値としている場合，リエントラントでないコードになります。これは，最後の区切り文字のアドレスをアプリケーション・プログラムの中で保存しておき，これを使って，*s* を空でない引数として渡すようにすることで回避できます。

strlen

文字列の長さを求めます。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
size_t  strlen(const char *s);
```

[戻り値]

終端を示す null 文字（¥0）の前に存在する文字の数を返します。

[詳細説明]

s の指す文字列の長さを求めます。

strerror

エラー番号の文字列変換を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char    *strerror(int errnum);
```

[戻り値]

変換した文字列へのポインタを返します。

[詳細説明]

処理系定義の対応関係にしたがって，エラー番号 *errnum* を文字列に変換します。*errnum* の値は，通常，グローバル変数 errno がコピーされたものです。指されている配列は，アプリケーション・プログラム側で変更しないでください。

7.4.4 メモリ管理関数

メモリ管理関数として，以下のものがあります。

表 7.6 メモリ管理関数

関数／マクロ名	概要
memchr	メモリ検索
memcmp	メモリ比較
memcpy	メモリ・コピー
memmove	メモリ移動
memset	メモリ・セット

memchr

メモリ検索を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void    *memchr(const void *s, int c, size_t length);
```

[戻り値]

c が見つかった場合はこの文字を指すポインタを返し，*c* が見つからなかった場合は null ポインタを返します。

[詳細説明]

s の指す領域の最初の *length* 個の文字の中に（char 型に変換された）文字 *c* が最初に現れた位置を求めます。

memcmp

メモリ比較を行います。

[所属]

標準ライブラリ

[指定形式]

#include <string.h>
int memcmp(const void **s1*, const void **s2*, size_t *n*);

[戻り値]

s1 の指すオブジェクトが *s2* の指すオブジェクトより大きい，等しい，または小さいかによって，0 より大きい，0 に等しい，または 0 より小さい整数を返します。

[詳細説明]

s1 の指すオブジェクトの最初の *n* 文字を *s2* の指すオブジェクトと比較します。

[使用例]

```
#include    <string.h>
int func(const void *s1, const void *s2) {
    int i;
    i = memcmp(s1, s2, 5);  /*s1 の指す文字列の最初の 5 文字を s2 の指す文字列の最初の 5 文字と比較
*/
    return(i);
}
```

memcpy

メモリ・コピーを行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void    *memcpy(void *out, const void *in, size_t n);
```

[戻り値]

out の値を返します。コピー元とコピー先の領域が重なっている場合，その動作は不定です。

[詳細説明]

n バイト分を *in* の指すオブジェクトから *out* の指すオブジェクトへコピーします。

memmove

メモリ移動を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void    *memmove(void *dst, void *src, size_t length);
```

[戻り値]

コピー先の *dst* の値を返します。

[詳細説明]

length 個の文字を，*src* の指すメモリ領域から *dst* の指すメモリ領域へ移動します。コピー元とコピー先の 2 つの領域が重なり合っていても，文字を *dst* の指すメモリ領域に正しくコピーします。

memset

メモリ・セットを行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void    *memset(const void *s, int c, size_t length);
```

[戻り値]

s の値を返します。

[詳細説明]

s の指すオブジェクトの最初の *length* 文字に unsigned char 型に変換した *c* の値をコピーします。

7.4.5 文字変換関数

文字変換関数として，以下のものがあります。

表 7.7 文字変換関数

関数／マクロ名	概要
toupper	英小文字から英大文字変換（引数が英大文字以外のときそのまま）
tolower	英大文字から英小文字変換（引数が英大文字以外のときそのまま）

toupper

英小文字から英大文字変換（引数が英小文字以外のときそのまま）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <ctype.h>
int toupper(int *c*);

[戻り値]

c に対して islower が真となる場合，それに対応して isupper が真となる文字を返します。そうでない場合，*c* を返します。

[詳細説明]

小文字の英字を対応する大文字の英字に変換し，他の文字はすべてそのままにするマクロです。
c が EOF ～ 255 の範囲の整数の場合にのみ定義されています。“#undef toupper” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

[使用例]

```
#include    <ctype.h>
int c = 'a';
int func() {
    int i;
    i = toupper(c); /*c の英小文字 'a' を英大文字 'A' に変換 */
    return(i);
}
```

tolower

英大文字から英小文字変換（引数が英大文字以外のときそのまま）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int tolower(int c);
```

[戻り値]

c に対して isupper が真となる場合，それに対応して islower が真となる文字を返します。そうでない場合，*c* を返します。

[詳細説明]

大文字の英字を対応する小文字の英字に変換し，他の文字はすべてそのままにするマクロです。
c が EOF ～ 255 の範囲の整数の場合にのみ定義されています。“#undef tolower” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

7.4.6 文字分類関数

文字分類関数として，以下のものがあります。

表 7.8 文字分類関数

関数／マクロ名	概要
isalnum	ASCII 英字，または数字であるかを判定
isalpha	ASCII 英字であるかを判定
isascii	ASCII コードであるかを判定
isupper	英大文字であるかを判定
islower	英小文字であるかを判定
isdigit	10 進数であるかを判定
isxdigit	16 進数であるかを判定
iscntrl	制御文字であるかを判定
ispunct	区切り文字であるかを判定
isspace	スペース／タブ／復帰／改行／垂直タブ／改ページであるかを判定
isprint	表示文字であるかを判定
isgraph	スペース以外の表示文字であるかを判定

isalnum

ASCII 英字，または数字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isalnum(int c);
```

[戻り値]

引数 *c* の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

ASCII 英字，または数字であるかどうか調べるマクロです。c が isascii で真になるか，または c が EOF の場合にのみ，定義されています。“#undef isalnum” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isalpha

ASCII 英字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isalpha(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

ASCII 英字であるかどうか調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isalpha” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isascii

ASCII コードであるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isascii(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

ASCII コード（0x00 ～ 0x7F）であるかどうかを調べるマクロです。すべての整数に対して定義されています。“#undef isascii” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isupper

英大文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isupper(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

大文字の英字（A ～ Z）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isupper” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

islower

英小文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int islower(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

小文字の英字（a ～ z）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef islower” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isdigit

10 進数であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isdigit(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

10 進数の数字であるかどうか調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isdigit” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isxdigit

16 進数であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isxdigit(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

16 進数の数字（0 ～ 9，a ～ f，または A ～ F）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isxdigit” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

iscntrl

制御文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int iscntrl(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

制御文字（0x00 ～ 0x1F，または 0x7F）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef iscntrl” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

ispunct

区切り文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int ispunct(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

印字可能な区切り文字 “isgraph (c) && ! isalnum (c)” であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef ispunct” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isspace

スペース／タブ／復帰／改行／垂直タブ／改ページであるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isspace(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

スペース，タブ，復帰，改行，垂直タブ，改ページ（0x09 ～ 0x0D，または 0x20）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isspace” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

isprint

表示文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

#include <ctype.h>
int isprint(int *c*);

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

表示文字（0x20 ～ 0x7E）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isprint” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

[使用例]

```
#include    <ctype.h>
void func(void) {
    int     i,j = 0;
    char    s[50];
        for(i =50; i <= 99; i++) {
            if(isprint(i)) {        /* コード 50 ～ 99 内の表示可能文字を配列 s に格納する */
                    s[j] = i;
                    j++;
            }
        }
    :
}
```

isgraph

スペース以外の表示文字であるかを判定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int isgraph(int c);
```

[戻り値]

c の値がそれぞれの記述に合致した場合（真）に 0 以外を返します。結果が偽であった場合は 0 を返します。

[詳細説明]

スペース（0x20）以外の表示文字[注]（0x20 ～ 0x7E）であるかどうかを調べるマクロです。*c* が isascii で真になるか，または *c* が EOF の場合にのみ，定義されています。“#undef isgraph” を使ってマクロ定義を無効にし，マクロ定義の代わりにコンパイル済みサブルーチンが使用できます。

注 printing character のことです。

7.4.7 標準入出力関数

標準入出力関数として，以下のものがあります。

表 7.9 標準入出力関数

関数／マクロ名	概要
fread	ストリームからの読み込み
getc	ストリームからの文字読み込み（fgetc と同じ）
fgetc	ストリームからの文字読み込み（getc と同じ）
fgets	ストリームからの一行読み込み
fwrite	ストリームへの書き込み
putc	ストリームへの文字書き込み（fputc と同じ）
fputc	ストリームへの文字書き込み（putc と同じ）
fputs	ストリームへの文字列出力
getchar	標準入力からの一文字読み込み
gets	標準入力からの文字列読み込み
putchar	標準出力ストリームへの文字書き込み
puts	標準出力ストリームへの文字列出力
sprintf	書式付き出力
fprintf	フォーマット指定したテキストをストリームへ出力
vsprintf	フォーマット指定したテキストを文字列へ書き込み
printf	フォーマット指定したテキストを標準出力ストリームへ出力
vfprintf	フォーマット指定したテキストをストリームへ書き込み
vprintf	フォーマット指定したテキストを標準出力ストリームへ書き込み
sscanf	書式付き入力
fscanf	ストリームからのデータ読み込みと解釈
scanf	標準入力ストリームからのテキストの読み込みと解釈
ungetc	入力ストリームへの文字押し戻し
rewind	ファイル位置指示子のリセット
perror	エラー処理

fread

ストリームからの読み込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
size_t fread(void **ptr*, size_t *size*, size_t *nmemb*, FILE **stream*);

[戻り値]

入力した要素数 *nmemb* を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す入力ストリームから，*size* の大きさの要素を *nmemb* 個入力し，*ptr* へ格納します。*stream* に指定できるのは，標準入出力の stdin だけです。

[使用例]

```
#include    <stdio.h>
void func(void) {
    struct {
        int     c;
        double  d;
    } buf[10];
    fread(buf, sizeof(buf[0]), sizeof(buf) / sizeof(buf [0]), stdin);
}
```

getc

ストリームからの文字読み込みを行います。（fgetc と同じ）

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int getc(FILE *stream);
```

[戻り値]

入力文字を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す入力ストリームから，1 文字を入力します。*stream* に指定できるのは，標準入出力の stdin だけです。

fgetc

ストリームからの文字読み込みを行います。（getc と同じ）

備考　　CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int fgetc(FILE **stream*);

[戻り値]

入力文字を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す入力ストリームから，1 文字を入力します。*stream* に指定できるのは，標準入出力の stdin だけです。

[使用例]

```
#include    <stdio.h>

int func(void) {
    int    c;
    c = fgetc(stdin);
    return(c);
}
```

fgets

ストリームからの一行読み込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
char    *fgets(char *s, int n, FILE *stream);
```

[戻り値]

s を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す入力ストリームから，最大 *n* - 1 文字を入力し，*s* へ格納します。文字の入力は，改行文字の検出によっても終了します。この場合，改行文字も *s* へ格納されます。最後に文字列の終結 null 文字が *s* へ格納されます。*stream* に指定できるのは，標準入出力の stdin だけです。

fwrite

ストリームへの書き込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
size_t  fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

[戻り値]

出力した要素数 *nmemb* を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す出力ストリームへ，*ptr* が指す配列から，*size* の大きさの要素を *nmemb* 個出力します。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。

putc

ストリームへの文字書き込みを行います。（fputc と同じ）

備考　　CS+ が提供するデバッグ機能では，サポートされていません。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

［戻り値］

文字 *c* を返します。
エラー・リターンはありません。

［詳細説明］

stream が指す出力ストリームへ，文字 *c* を出力します。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。

fputc

ストリームへの文字書き込みを行います。（putc と同じ）

備考　　CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int fputc(int *c*, FILE **stream*);

[戻り値]

文字 *c* を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す出力ストリームへ，文字 *c* を出力します。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。

[使用例]

```
#include    <stdio.h>
void func(void) {
    fputc('a', stdout);
}
```

fputs

ストリームへの文字列出力を行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

[戻り値]

0 を返します。
エラー・リターンはありません。

[詳細説明]

stream が指す出力ストリームへ，文字列 *s* を出力します。文字列の終端 null 文字は出力しません。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。

getchar

標準入力からの一文字読み込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int getchar(void);
```

[戻り値]

入力文字を返します。
エラー・リターンはありません。

[詳細説明]

標準入出力の stdin から，1 文字を入力します。

gets

標準入力からの文字列読み込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
char    *gets(char *s);
```

[戻り値]

s を返します。
エラー・リターンはありません。

[詳細説明]

標準入出力の stdin から，改行文字を検出するまで文字を入力し，*s* へ格納します。入力した改行文字は捨て，最後に，文字列の終結 null 文字が *s* へ格納されます。

putchar

標準出力ストリームへの文字書き込みを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int putchar(int c);
```

[戻り値]

文字 *c* を返します。
エラー・リターンはありません。

[詳細説明]

標準入出力の stdout へ，文字 *c* を出力します。

puts

標準出力ストリームへの文字列出力を行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int puts(const char *s);
```

[戻り値]

0 を返します。
エラー・リターンはありません。

[詳細説明]

標準入出力の stdout へ，文字列 *s* を出力します。文字列の終端 null 文字は出力せず，代わりに改行文字を出力します。

sprintf

書式付き出力を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int sprintf(char *s, const char *format[, arg, ...]);
```

[戻り値]

出力された文字（null 文字（¥0）は除きます）の数を返します。
エラー・リターンはありません。

[詳細説明]

それぞれの *arg* に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを *s* の指す配列に書き出します。
書式に対して引数が十分にない場合，動作は不定です。書式文字列の終わりに到達するとリターンします。書式で必要としている以上に引数がある場合，余分の引数を無視します。また，*s* の領域が引数の 1 つと重なっていると動作は不定になります。
format は，“ 後ろに続く引数がどのような出力に変換されるか ” を指定しています。書き込まれた文字の最後には null 文字（¥0）が付加されます（null 文字（¥0）は返り値におけるカウントの対象とはなりません）。
format は，次に示す 2 種類のディレクティブにより構成されます。

通常文字	変換されずにそのまま出力にコピーされるものです（“%” 以外）。
変換指示	0 個以上の引数を取り込み，指示を与えるものです。

各変換指示は，文字 “%” で始まります（出力中に “%” を入れたい場合は，書式文字列の中では “%%” とします）。“%” の後ろは，次のようになります。

%[フラグ][フィールド長][精度][サイズ][型指定文字]

それぞれの変換指示について，次に説明します。

(1) フラグ
任意の順に置かれた，変換指示の意味を修飾する 0 個以上のフラグです。フラグ文字とその意味を次に示します。

-	変換された結果をフィールド中に左詰めにし，右側は空白で満たされます（このフラグが指定されない場合，変換された結果は右詰めにされます）。
+	符号付きの変換の結果を常に + 符号，または - 符号で始めます（このフラグが指定されない場合，変換された結果は，負の値が変換された場合にのみ符号で始められます）。
スペース	符号付きの変換の最初の文字が符号でない場合，または符号付きの変換が文字を生じない場合，その結果の前にスペース（“ ”）を付けます。スペース・フラグと + フラグとが両方現れる場合，スペース・フラグは無視されます。

#	結果を " 別の形式注 1" に変換します。o 変換に対しては，その変換結果の最初の数字が 0 になるようにその精度を増やします。x，または X 変換に対しては，0 以外の変換結果の先頭に 0x，または 0X を付加します。e，f，g，E，G 変換に対しては，その変換結果に小数点以下の数字が存在しない場合であっても，小数点 "." を付加します注 2。g，G 変換に対しては，変換結果から後ろに続く 0 が削除されないようにします。これら以外の変換に対しては，その動作は不定となります。
0	d，e，f，g，i，o，u，x，E，G，X 変換に対し，フィールド長を埋めるために，符号，または基底の指示に続いて 0 を付加します。 0 フラグと - フラグの両方が指定された場合，0 フラグは無視されます。d，i，o，u，x，X 変換については，精度を指定している場合，ゼロ（0）フラグを無視します。 0 はフラグとして解釈され，フィールド幅の始まりとは解釈されないことに注意してください。 これら以外の変換に対してはその動作は不定となります。

注 1. alternate format のことです。

注 2. 通常，小数点は，その後ろに数字が続く場合にのみ現れます。

(2) フィールド長

オプショナルな最小フィールド長です。変換された値がこのフィールド長より小さい場合，左側にスペースが詰められます（前述の左詰めフラグが与えられた場合は右側にスペースが詰められます）。このフィールド長は "*"，または 10 進整数の形を取ります。"*" で指定した場合，int 型の引数をフィールド長として使用します。負のフィールド長は，サポートしていません。負のフィールド長を指定しようとすると，正のフィールド長の前にマイナス（-）フラグが付いたものと解釈されます。

(3) 精度注

これに与えられる値は，d，i，o，u，x，X 変換に対しては現れる数字の個数の最小値であり，e，f，E 変換に対しては "." の後ろに現れる数字の個数であり，g，G 変換に対しては最大有効桁数です。精度は，"*"，または 10 進整数が後ろに続く "." の形式を取ります。"*" を指定した場合，int 型の引数を精度として使用します。負の精度を指定した場合，精度を省略したものとみなされます。"." のみが指定された場合，精度は 0 とされます。精度がこれら以外の変換指示とともに現れた場合，動作は不定となります。

注 precision のことです。

(4) サイズ

対応する引数のデータ型を解釈するためのデフォルトの方法を変更する，任意選択のサイズ文字 h，l，ll，および L です。

h を指定した場合，後ろに続く d，i，o，n，u，x，X の型指定を強制的に short，または unsigned short に適用します。

l を指定した場合，後ろに続く d，i，o，u，x，X の型指定を強制的に long，または unsigned long に適用します。l はさらに，後ろに続く n の型指定を強制的に long へのポインタに適用します。h，または l といっしょにこれと別の型指定文字を使用した場合，その動作は不定です。

ll を指定した場合，後ろに続く d，i，o，u，x，X の型指定を強制的に long long，または unsigned long long に適用します。ll はさらに，後ろに続く n の型指定を強制的に long long へのポインタに適用します。ll と一緒にこれ以外の型指定文字を使用した場合，その動作は不定です。

L を指定した場合，後ろに続く e，E，f，g，G の型指定を強制的に long double に適用します。L といっしょにこれ以外の型指定文字を使用した場合，その動作は不定です。

(5) 型指定文字

適用される変換の型を指定する文字です。

変換の型を指定する文字とその意味を次に示します。

%	文字 "%" を出力します。引数は変換されません。変換指示は "%%" となります。
c	int 型の引数を unsigned char 型に変換し，変換結果の文字を出力します。
d	int 型の引数を符号付きの 10 進数に変換します。
e，E	double 型の引数を，小数点の前に（引数が 0 でない場合 0 でない）1 つの文字を持ち，小数点以下の数字の個数は精度に等しい [-]d.ddddde±dd の形式に変換します。E 変換指示は，指数部が "e" ではなく "E" で始まる数字を生成します。
f	double 型の引数を [-]dddd.dddd の形式の 10 進表記に変換します。
g，G	精度には仮数部の数字の個数を指定するものとし，double 型の引数を e（G 変換指示の場合 E），または f の形式に変換します。変換結果の末尾の 0 は結果の小数点部から除かれます。小数点は，後ろに数字が続く場合にのみ現れます。

i	d の変換と同じ変換をします。
n	同じオブジェクト内で出力された文字の個数を格納します。int 型へのポインタを引数とします。
p	処理系定義書式でポインタを出力します。CC-RH では，ポインタを unsigned long として扱っています（lu の指定と同じです）。
o，u，x，X	unsigned int 型の引数を dddd の形式の 8 進表記（o），符号なしの 10 進表記（u），符号なしの 16 進表記（x，または X）に変換します。x 変換に対しては文字 abcdef が用いられ X 変換に対しては文字 ABCDEF が用いられます。
s	引数は文字型の配列を指すポインタでなければなりません。この配列からの文字を，終端を示す null 文字（¥0）の前まで（null 文字（¥0）自身は含まずに）出力します。精度が指定された場合，それ以上の個数の文字は出力されません。精度が指定されなかった，または精度がこの配列の大きさ以上の値であった場合，この配列は null 文字（¥0）を含むようにしてください。

［使用例］

```
#include    <stdio.h>
void func(int val) {
  char    s[20];
  sprintf(s, "%-10.51x\n", val);  /*val の値に対し，左詰め，フィールド長 10，精度 5，サイズ
long，*/
                                  /*16 進表記を指定し，改行文字を付加して s の指す配列へ出力 */
}
```

fprintf

フォーマット指定したテキストをストリームへ出力します。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int fprintf(FILE **stream*, const char **format*[, *arg*, ...]);

[戻り値]

出力された文字数を返します。

[詳細説明]

それぞれの *arg* に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを *stream* に出力します。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。*format* の記述方法は sprintf と同様です。sprintf と違って，最後に null 文字（¥0）は出力されません。

[注意事項]

stdout には I/O アドレスなど 1 メモリ・アドレスを割り当てます。デバッガとの連携で stdout を使用するには，stdio.h ファイルで定義されている，ストリーム構造体の初期値設定が必要です。関数を呼び出す前に，初期値設定を行ってください。

【stdio.h におけるストリーム構造体の定義】

```
typedef struct {
        int             mode;           /*with error descriptions*/
        unsigend        handle;
        int             unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())
#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

構造体の第一メンバ mode は，入出力状態を示します。ACCSD_OUT/ACCSD_IN として内部定義されています。第三メンバ unget_c は，押し戻し文字（stdin のみ）を示し，-1 として内部定義されています。
-1 の場合，押し戻し文字 “ なし ” を表します。第二メンバ handle は，入出力 I/O アドレスを示します。handle には，使用するデバッガで決められている値を設定してください。

【入出力 I/O アドレス設定例】

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

［使用例］

```
#include    <stdio.h>
void func(int val) {
    fprintf(stdout, "%-10.5x\n", val);
}
/* 汎用のエラー報告ルーチンにおける使用例 */
void error(char *function_name, char *format, ...) {
    va_list arg;
    va_start(arg, format);
    fprintf(stderr, "ERROR in %s:", function_name); /* エラーが発生した関数名を出力 */
    vfprintf(stderr, format, arg);                  /* 残りのメッセージを出力 */
    va_end(arg);
}
```

vsprintf

フォーマット指定したテキストを文字列へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int vsprintf(char *s, const char *format, va_list arg);
```

[戻り値]

出力された文字（null 文字（¥0）は除きます）の数を返します。
エラー・リターンはありません。

[詳細説明]

arg の指す引数列に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを *s* が指す配列に出力します。本関数は，可変個数実引数並びを *arg* で置き換えた sprintf と等価です。本関数の呼び出しの前に，va_start で *arg* を初期化しておく必要があります。

printf

フォーマット指定したテキストを標準出力ストリームへ出力します。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int printf(const char **format*[, *arg*, ...]);

[戻り値]

出力された文字数を返します。

[詳細説明]

それぞれの *arg* に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを標準入出力の stdout に出力します。*format* の記述方法は sprintf と同様です。sprintf と異なり，最後に null 文字（¥0）は出力されません。

[注意事項]

stream に stdout（標準出力），stderr（標準エラー）を指定します。ストリームの入出力先は I/O アドレスなど 1 メモリ・アドレスを割り当てます。デバッガとの連携でこれらのストリームを使用するには，stdio.h ファイルで定義されている，ストリーム構造体の初期値設定が必要です。関数を呼び出す前に，初期値設定を行ってください。

【stdio.h におけるストリーム構造体の定義】

```
typedef struct {
        int             mode;           /*with error descriptions*/
        unsigend        handle;
        int             unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())
#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

構造体の第一メンバ mode は，入出力状態を示します。ACCSD_OUT/ACCSD_IN として内部定義されています。第三メンバ unget_c は，押し戻し文字（stdin のみ）を示し，-1 として内部定義されています。
-1 の場合，押し戻し文字 “ なし ” を表します。第二メンバ handle は，入出力 I/O アドレスを示します。handle には，使用するデバッガで決められている値を設定してください。

【入出力 I/O アドレス設定例】

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

vfprintf

フォーマット指定したテキストをストリームへ書き込みます。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int vfprintf(FILE **stream*, const char **format*, va_list *arg*);

[戻り値]

出力された文字数を返します。

[詳細説明]

arg の指す引数列に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを *stream* に出力します。*stream* に指定できるのは，標準入出力の stdout と stderr だけです。*format* の記述方法は sprintf と同様です。本関数は，可変個数実引数並びを *arg* で置き換えた fprintf と等価です。本関数の呼び出しの前に，va_start で *arg* を初期化しておく必要があります。

[注意事項]

stream に stdout（標準出力），stderr（標準エラー）を指定します。ストリームの入出力先は I/O アドレスなど 1 メモリ・アドレスを割り当てます。デバッガとの連携でこれらのストリームを使用するには，stdio.h ファイルで定義されている，ストリーム構造体の初期値設定が必要です。関数を呼び出す前に，初期値設定を行ってください。

【stdio.h におけるストリーム構造体の定義】

```
typedef struct {
        int             mode;           /*with error descriptions*/
        unsigend        handle;
        int             unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())
#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

構造体の第一メンバ mode は，入出力状態を示します。ACCSD_OUT/ACCSD_IN として内部定義されています。第三メンバ unget_c は，押し戻し文字（stdin のみ）を示し，-1 として内部定義されています。
-1 の場合，押し戻し文字 “ なし ” を表します。第二メンバ handle は，入出力 I/O アドレスを示します。handle には，使用するデバッガで決められている値を設定してください。

【入出力 I/O アドレス設定例】

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

［使用例］

```
#include    <stdio.h>
void func(int val) {
    fprintf(stdout, "%-10.5x\n", val);
}
/* 汎用のエラー報告ルーチンにおける使用例 */
void error(char *function_name, char *format, ...) {
    va_list arg;
    va_start(arg, format);
    fprintf(stderr, "ERROR in %s:", function_name); /* エラーが発生した関数名を出力 */
    vfprintf(stderr, format, arg);                  /* 残りのメッセージを出力 */
    va_end(arg);
}
```

vprintf

フォーマット指定したテキストを標準出力ストリームへ書き込みます。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int vprintf(const char **format*, va_list *arg*);

[戻り値]

出力された文字数を返します。

[詳細説明]

arg の指す引数列に *format* の指す文字列で指定された書式を適用し，それにより出力された書式付きデータを標準入出力の stdout に出力します。*format* の記述方法は sprintf と同様です。本関数は，可変個数実引数並びを *arg* で置き換えた printf と等価です。本関数の呼び出しの前に，va_start で *arg* を初期化しておく必要があります。

[注意事項]

stream に stdout（標準出力），stderr（標準エラー）を指定します。ストリームの入出力先は I/O アドレスなど 1 メモリ・アドレスを割り当てます。デバッガとの連携でこれらのストリームを使用するには，stdio.h ファイルで定義されている，ストリーム構造体の初期値設定が必要です。関数を呼び出す前に，初期値設定を行ってください。

【stdio.h におけるストリーム構造体の定義】

```
typedef struct {
        int             mode;           /*with error descriptions*/
        unsigend        handle;
        int             unget_c;
} FILE;
typedef int     fpos_t;

extern  FILE*   _REL_stdin();
extern  FILE*   _REL_stdout();
extern  FILE*   _REL_stderr();
#define stdin   (_REL_stdin())
#define stdout  (_REL_stdout())
#define stderr  (_REL_stderr())
```

構造体の第一メンバ mode は，入出力状態を示します。ACCSD_OUT/ACCSD_IN として内部定義されています。第三メンバ unget_c は，押し戻し文字（stdin のみ）を示し，-1 として内部定義されています。
-1 の場合，押し戻し文字 “ なし ” を表します。第二メンバ handle は，入出力 I/O アドレスを示します。handle には，使用するデバッガで決められている値を設定してください。

【入出力 I/O アドレス設定例】

```
stdout->handle = 0xfffff000;
stderr->handle = 0x00fff000;
stdin->handle  = 0xfffff002;
```

sscanf

書式付き入力を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int sscanf(const char *s, const char *format[, arg, ...]);
```

[戻り値]

走査，変換，格納が正常に実行できた入力フィールドの個数を返します。返却値には，格納されなかった走査済みフィールドは含まれません。ファイルの終わりで読み込もうとした場合，返却値は EOF です。フィールドが格納されなかった場合は，返却値は 0 です。

[詳細説明]

format の指す文字列で指定された書式に従い，その後ろに続く引数 *arg* を，変換された入力を格納するオブジェクトを指すポインタとして扱い，*s* の指す配列から変換する入力を読み込みます。

format には，認識されうる入力列，および " 代入のためにどのように変換を行うか " ということを指定します。*format* に対し十分な引数が存在しない場合，その動作は不定となります。引数が残っているのに *format* が使い果たされた場合，残された引数は無視されます。

format は，次に示す 3 種類のディレクティブにより構成されます。

1 個以上の空白類	スペース（ ），タブ（¥t），改行（¥n）です。 本関数を実行して，文字列内に空白文字が見つかった場合，次の空白でない文字まで連続するすべての空白文字を読み込みます（格納はしません）。
通常の文字	"%" 以外のすべての ASCII 文字です。 本関数を実行して，文字列内に通常の文字が見つかった場合，それを読み込みますが，格納はしません。変換指示により，本関数は，入力フィールドから文字列を読み込み，特定の型の値に変換し，引数で指定した位置に格納します。変換指示で明示されて一致しているのでなければ，後ろに続く空白は読み込まれません。
変換指示	0 個以上の引数を取り込み，変換の指示を与えます。

各変換指示は "%" で始まります。"%" の後ろは，次のようになります。

%[代入抑制文字][フィールド長][サイズ][型指定文字]

それぞれの変換指示について，次に説明します。

(1) 代入抑制文字
入力フィールドの解釈，および代入を抑制する "*" です。

(2) フィールド長
最大フィールド長を規定する 0 以外の 10 進整数です。入力フィールドを変換する前に読み込まれる最大文字数を指定します。入力フィールドがこのフィールド長より小さい場合，本関数はフィールド内のすべての文字を読み込み，次のフィールドとその変換指示へ進みます。また，フィールド長分を読み込む前に，空白文字，または変換できない文字が見つかった場合，その文字までの文字群を読み込み，変換し，格納します。その後，本関数は次の変換指示へ進みます。

(3) サイズ
対応する引数のデータ型を解釈するデフォルトの方法を変更する，任意選択のサイズ文字 h，l，ll，および L です。

h を指定した場合，後ろに続く d，i，n，o，u，x の型指定を強制的に short int 型に変換し，short 型で格納します。c，e，f，n，p，s，D，I，O，U，X では，何もしません。
l を指定した場合，後ろに続く d，i，n，o，u，x の型指定を強制的に long int 型に変換し，long 型で格納します。e，f，g では，強制的に double 型に変換し，double 型で格納します。c，n，p，s，D，I，O，U，X では，何もしません。
ll を指定した場合，後ろに続く d，i，o，u，x，X の型指定を強制的に long long 型に変換し，long long 型で格納します。他の型指定では，何もしません。
L を指定した場合，後ろに続く e，f，g の型指定を強制的に long double 型に変換し，long double 型で格納します。他の型指定では，何もしません。
これら以外の場合，その動作は不定です。

(4) 型指定文字
適用される変換の型を指定する文字です。変換の型を指定する文字とその意味を次に示します。

%	文字 “%” にマッチします。変換も代入も行われません。変換指示は “%%” となります。
c	1 文字を走査します。対応する引数は “char *arg” にしてください。
d	10 進整数を対応する引数に読み込みます。対応する引数は “int *arg” にしてください。
e，f，g	浮動小数点数を対応する引数に読み込みます。対応する引数は “float *arg” に，サイズ指定 l を指定した場合は “double *arg” にしてください。
i	10 進，8 進，または 16 進整数を対応する引数に読み込みます。対応する引数は “int *arg” にしてください。
n	対応する引数に読み込んだ文字の個数を格納します。対応する引数は “int *arg” にしてください。
o	8 進整数を対応する引数に読み込みます。対応する引数は “int *arg” にしてください。
p	走査したポインタを格納します。これは処理系定義です。 CC-RH では，%p を %U とまったく同じように処理しています。対応する引数は “void **arg” にしてください。
s	与えられた配列の中に文字列を読み込みます。対応する引数は “char arg[ ]” にしてください。
u	符号なし 10 進整数を対応する引数に読み込みます。対応する引数は “unsigned int *arg” にしてください。
x，X	16 進整数を対応する引数に読み込みます。対応する引数は “int *arg” にしてください。
D	10 進整数を対応する引数に読み込みます。対応する引数は “long *arg” にしてください。
E，F，G	浮動小数点数を対応する引数に読み込みます。対応する引数は “float *arg” に，サイズ指定 l を指定した場合は “double *arg” にしてください。
I	10 進，8 進，または 16 進整数を対応する引数に読み込みます。対応する引数は “long *arg” にしてください。
O	8 進整数を対応する引数に読み込みます。対応する引数は “long *arg” にしてください。
U	符号なし 10 進整数を対応する引数に読み込みます。対応する引数は “unsigned long *arg” にしてください。

[ ]	空でない文字列を引数 arg で始まるメモリの中へ読み込みます。この領域には，文字列と，自動的に付加される，文字列の終わりを示す null 文字（\0）とを受け入れられる大きさが必要です。対応する引数は “char *arg” にしてください。 [ ] で囲まれた文字パターンを，型指定文字 s の代わりに使用することができます。文字パターンは，本関数の入力フィールドを構成する文字の検索セットを定義する文字集合です。[ ] 内の最初の文字が “^” の場合，検索セットは反転され，[ ] 内の文字以外のすべての ASCII 文字が含まれます。また，ショートカットとして使用できる範囲指定機能もあります。たとえば，%[0-9] は，すべての 10 進数字と一致します。この集合内では，“-” は最初，または最後の文字にはできません。“-” の前の文字は，その後ろの文字よりも辞書式順序で小さくなるようにしてください。 - %[abcd] a，b，c，d のみを含む文字列と一致します。 - %[^abcd] a，b，c，d 以外の任意の文字を含む文字列と一致します。 - %[A-DW-Z] A，B，C，D，W，X，Y，Z を含む文字列と一致します。 - %[z-a] z，-，a と一致します（範囲指定とはみなされません）。

浮動小数点数（型指定文字 e，f，g，E，F，G）の場合，次の一般形式に対応させてください。

[+ | -] ddddd [.] ddd [E | e [+ | -] ddd]

ただし，上記の一般形式のうち [ ] で囲まれた部分は任意選択であり，ddd は 10 進数字を表します。

[注意事項]

- 通常のフィールド終了文字に到達する前に，特定フィールドの走査を停止したり完全に終了したりする可能性があります。
- 次の状況では，その時点でのフィールドの走査，格納を停止し，次の入力フィールドに移動します。
  - 代入抑制文字（*）が書式指定の中で “%” の後ろに現れており，その時点の入力フィールドは走査されているが格納はされていない。
  - フィールド長（正の 10 進整数）指定文字を読み込んだ。
  - 読み込む次の文字がその変換指示では変換できない（たとえば，指示が 10 進のときに Z を読み込む場合）。
  - 入力フィールド内の次の文字が検索セット内に現れていない（または反転検索セット内に現れている）。

  以上の理由からその時点の入力フィールドの走査を停止すると，次の文字が未読であるとみなされ，次の入力フィールドの最初の文字，またはその入力のあとの読み込み操作の最初の文字として使用されます。
- 本関数は，次の状況では終了します。
  - 入力フィールド内の次の文字が変換する文字列内の対応する通常文字と一致していない。
  - 入力フィールド内の次の文字が EOF である。
  - 変換する文字列が終了した。
- 変換する文字列に変換指示の一部ではない文字の並びが含まれている場合，この同じ文字の並びは入力の中に現れないようにしてください。本関数は一致する文字を走査しますが，格納はしません。不一致があった場合，一致していない最初の文字は読み取られていなかったかのように入力の中に残っています。

[使用例]

```
#include    <stdio.h>
void func(void) {
    int         i, n;
    float       x;
    const char  *s;
    char        name[10];
    s = "23 11.1e-1 NAME";
    n = sscanf(s,"%d%f%s", &i, &x, name);    /*i に 23，x に 1.110000，name に “NAME” を格納，
*/
                                             /* 戻り値 n は 3*/
}
```

fscanf

ストリームからのデータ読み込みと解釈を行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format[, arg, ...]);
```

[戻り値]

走査，変換，格納が正常に実行できた入力フィールドの個数を返します。返却値には，格納されなかった走査済みフィールドは含まれません。ファイルの終わりで読み込もうとした場合，返却値は EOF です。フィールドが格納されなかった場合は，返却値は 0 です。

[詳細説明]

format の指す文字列で指定された書式に従い，その後ろに続く引数 *arg* を，変換された入力を格納するオブジェクトとして扱い，*stream* から変換する入力を読み込みます。*stream* に指定できるのは，標準入出力の stdin だけです。*format* の記述方法は sscanf と同様です。

scanf

標準出力ストリームからのテキストの読み込みと解釈を行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

#include <stdio.h>
int scanf(const char **format*[, *arg*, ...]);

[戻り値]

走査，変換，格納が正常に実行できた入力フィールドの個数を返します。返却値には，格納されなかった走査済みフィールドは含まれません。ファイルの終わりで読み込もうとした場合，返却値は EOF です。フィールドが格納されなかった場合は，返却値は 0 です。

[詳細説明]

format の指す文字列で指定された書式に従い，その後ろに続く引数 *arg* を，変換された入力を格納するオブジェクトとして扱い，標準入出力の stdin から変換する入力を読み込みます。*format* の記述方法は sscanf と同様です。

[使用例]

```
#include    <stdio.h>
void func(void) {
    int     i, n;
    double  x;
    char    name[10];
    n = scanf("%d%lf%s", &i, &x, name); /*“23 11.1e-1 NAME” の形式の stdin から入力を */
                                        /* 書式化入力 */
}
```

ungetc

入力ストリームへの文字押し戻しを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

[戻り値]

文字 *c* を返します。
エラー・リターンはありません。

[詳細説明]

文字 *c* を *stream* が指す入力ストリームへ押し戻します。ただし，*c* が EOF の場合，押し戻しは行われません。
押し戻された文字 *c* は，次の文字入力の際，最初の文字として入力されることになります。本関数によって，押し戻すことができるのは 1 文字だけです。本関数を続けて実行した場合，効果があるのは最後だけです。*stream* に指定できるのは，標準入出力の stdin だけです。

rewind

ファイル位置指示子のリセットを行います。

備考 CS+ が提供するデバッグ機能では，サポートされていません。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
void    rewind(FILE *stream);
```

[詳細説明]

stream が指す入力ストリームのエラー表示子をクリアし，ファイル位置表示子をファイルの先頭に位置付けます。

ただし，*stream* に指定できるのは，標準入出力の stdin だけです。そのため，本関数は ungetc による押し戻し文字を破棄する効果だけを持ちます。

perror

エラー処理を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
void    perror(const char *s);
```

[詳細説明]

グローバル変数 errno に対応するエラー・メッセージを stderr へ出力します。
出力されるメッセージは，次のようになります。

s が NULL でない場合	`fprintf(stderr, "%s:%s\n", s, s_fix);`
s が NULL の場合	`fprintf(stderr, "%s\n", s_fix);`

s_fix は，次のようになります。

errno が EDOM の場合	"EDOM error"
errno が ERANGE の場合	"ERANGE error"
errno が 0 の場合	"no error"
その他の場合	"error *xxx*"（*xxx* は abs (errno) % 1000）

[使用例]

```
#include    <stdio.h>
#include    <errno.h>
void func(double x) {
    double  d;
    errno = 0;
    d = exp(x);
    if(errno)
        perror("func");     /*exp で演算例外が発生した場合，perror を呼び出す */
}
```

7.4.8 標準ユーティリティ関数

標準ユーティリティ関数として，以下のものがあります。

表 7.10 標準ユーティリティ関数

関数／マクロ名	概要
abs	絶対値（int 型）を出力
labs	絶対値（long 型）を出力
llabs	絶対値（long long 型）を出力
bsearch	バイナリ検索
qsort	整列
div	除算（int 型）
ldiv	除算（long 型）
lldiv	除算（long long 型）
atoi	文字列を整数（int 型）へ変換
atol	文字列を整数（long 型）へ変換
atoll	文字列を整数（long long 型）へ変換
strtol	文字列を整数（long 型）へ変換し，最終文字列へのポインタを格納
strtoul	文字列を整数（unsigned long 型）へ変換し，最終文字列へのポインタを格納
strtoll	文字列を整数（long long 型）へ変換し，最終文字列へのポインタを格納
strtoull	文字列を整数（unsigned long long 型）へ変換し，最終文字列へのポインタを格納
atoff	文字列を浮動小数点数（float 型）へ変換
atof	文字列を浮動小数点数（double 型）へ変換
strtodf	文字列を浮動小数点数（float 型）へ変換（最終文字列へのポインタ格納）
strtod	文字列を浮動小数点数（double 型）へ変換（最終文字列へのポインタ格納）
calloc	メモリ割り当て（ゼロ初期化付き）
malloc	メモリ割り当て（ゼロ初期化なし）
realloc	メモリの再割り当て
free	メモリ開放
rand	疑似乱数列生成
srand	疑似乱数列の種類を設定
abort	プログラムを異常終了する

abs

絶対値（int 型）を出力します。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
int abs(int *j*);

[戻り値]

j の絶対値（*j* の大きさ），|*j*| を返します。

[詳細説明]

j の絶対値（*j* の大きさ），|*j*| を求めます。つまり，*j* が負の数の場合，結果は *j* の反転であり，負でない場合，*j* となります。

[使用例]

```
#include    <stdlib.h>
void func(int l) {
    int val;
    val = -15;
    l = abs(val);    /*val の値の絶対値 15 を l に返す */
}
```

labs

絶対値（long 型）を出力します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long    labs(long j);
```

[戻り値]

j の絶対値（*j* の大きさ），| *j* | を返します。

[詳細説明]

j の絶対値（*j* の大きさ），| *j* | を求めます。つまり，*j* が負の数の場合，結果は *j* の反転であり，負でない場合，*j* となります。abs と同じですが，int 型の値の代わりに long 型を使用し，戻り値も long 型です。

llabs

絶対値（long long 型）を出力します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long long      llabs(long long j);
```

[戻り値]

j の絶対値（*j* の大きさ），| *j* | を返します。

[詳細説明]

j の絶対値（*j* の大きさ），| *j* | を求めます。つまり，*j* が負の数の場合，結果は *j* の反転であり，負でない場合，*j* となります。abs と同じですが，int 型の値の代わりに long long 型を使用し，戻り値も long long 型です。

[注意事項]

-Xansi オプション指定時の動作結果は不定です。

bsearch

バイナリ検索を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
void* bsearch(const void **key*, const void **base*, size_t *nmemb*, size_t *size*, int (**compar*)(const void *, const void*));

[戻り値]

key と一致する配列の要素へのポインタを返します。一致する要素が複数ある場合，結果はその中で最初に見つかった要素を指します。*key* と一致する要素が見つからなかった場合，null ポインタを返します。

[詳細説明]

バイナリ検索法により，*base* から始まる配列の中で，*key* と一致する要素を検索します。*nmemb* は，配列の要素数です。*size* は，各要素のサイズです。配列は，*compar*（最後の引数）が指す比較関数に関し昇順で整列するようにしてください。*compar* が指す比較関数は，2 つの引数を持つように定義してください。結果は，1 番目の引数が 2 番目の引数よりも小さい場合は負，2 つの引数が一致する場合はゼロ，1 番目の引数が 2 番目の引数よりも大きい場合は正の整数を返すようにしてください。

[使用例]

```
#include    <stdlib.h>
#include    <string.h>
int compar(const void *x, const void *y);

void func(void) {
    static  char    *base[] = {"a", "b", "c", "d", "e", "f"};
    char            *key = "c";     /* 検索キーは “c”*/
    char            **ret;
                                    /*ret に “c” へのポインタを格納 */
    ret = (char **) bsearch((char *) &key, (char *) base, 6, sizeof(char *), compar);
}
int compar(const void *x, const void *y {
    return(strcmp(x, y));           /* 引数を比較して正，ゼロ，または負の整数を返す */
}
```

qsort

整列します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void    qsort(void *base, size_t nmemb, size_t size, int (*compar)(const void*, const void *));
```

[詳細説明]

base の指す配列を *compar* が指す比較関数に関し昇順に整列します。*nmemb* は配列の要素数，*size* は各要素のサイズです。*compar* が指す比較関数は bsearch と同様です。

div

除算（int 型）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
div_t div(int *n*, int *d*);

[戻り値]

除算の結果を格納した構造体を返します。

[詳細説明]

int 型の値を除算する場合に使用します。
分子 *n* を分母 *d* で割ったその商と剰余を算出し，その 2 つの整数を次に示す構造体 div_t のメンバとして格納します。

```
typedef struct {
    int quot;
    int rem;
} div_t;
```

quot は商で，rem は剰余です。*d* がゼロでない場合，“*r* = div (*n*, *d*);” であれば，*n* は “*r*.rem + *d* * *r*.quot” に等しい値です。
d がゼロの場合，結果の quot メンバは，符号が *n* と同じで，大きさが表現可能な最大の大きさとなります。また，rem メンバは 0 です。

[使用例]

```
#include    <stdlib.h>
void func(void) {
    div_t   r;
    r = div(110, 3);    /*r.quot には 36，r.rem には 2 を格納 */
}
```

ldiv

除算（long 型）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
ldiv_t ldiv(long *n*, long *d*);

[戻り値]

除算の結果を格納した構造体を返します。

[詳細説明]

long 型の値を除算する場合に使用します。
分子 *n* を分母 *d* で割ったその商と剰余を算出し，その 2 つの整数を次に示す構造体 ldiv_t のメンバとして格納します。

```
typedef struct {
    long    quot;
    long    rem;
} ldiv_t;
```

quot は商で，rem は剰余です。*d* がゼロでない場合，“*r* = ldiv (*n*, *d*);” であれば，*n* は “*r*.rem + *d* * *r*.quot” に等しい値です。
d がゼロの場合，結果の quot メンバは，符号が *n* と同じで，大きさが表現可能な最大の大きさとなります。また，rem メンバは 0 です。

lldiv

除算（long long 型）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
lldiv_t lldiv(long long *n*, long long *d*);

[戻り値]

除算の結果を格納した構造体を返します。

[詳細説明]

long long 型の値を除算する場合に使用します。
分子 *n* を分母 *d* で割ったその商と剰余を算出し，その 2 つの整数を次に示す構造体 lldiv_t のメンバとして格納します。

```
typedef struct {
    long long   quot;
    long long   rem;
} lldiv_t;
```

quot は商で，rem は剰余です。*d* がゼロでない場合，“*r* = lldiv (*n*, *d*);” であれば，*n* は “*r*.rem + *d* * *r*.quot” に等しい値です。
d がゼロの場合，結果の quot メンバは，符号が *n* と同じで，大きさが表現可能な最大の大きさとなります。また，rem メンバは 0 です。

[注意事項]

-Xansi オプション指定時の動作結果は不定です。

atoi

文字列を整数（int 型）へ変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
int atoi(const char *str);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。

[詳細説明]

str の指す文字列の最初の部分を int 型の表現に変換します。本関数は，“(int) strtol (*str*, NULL, 10)” と同じです。

atol

文字列を整数（long 型）へ変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long    atol(const char *str);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。

[詳細説明]

str の指す文字列の最初の部分を long int 型の表現に変換します。本関数は，“strtol (*str*, NULL, 10)” と同じです。

atoll

文字列を整数（long long 型）へ変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long long   atoll(const char *str);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。

[詳細説明]

str の指す文字列の最初の部分を long long int 型の表現に変換します。本関数は，“strtoll (*str*, NULL, 10)” と同じです。

[注意事項]

-Xansi オプション指定時の動作結果は不定です。

strtol

文字列を整数（long 型）へ変換し，最終文字列へのポインタを格納します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long    strtol(const char *str, char **ptr, int base);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる（変換された値が大きすぎる）場合，LONG_MAX，または LONG_MIN を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

str の指す文字列の最初の部分を long 型の表現に変換します。まず，入力文字を次の 3 つの部分，“ 最初の空白類 ”，“*base* の値により定められる基数において表現され，整数にする対象となる列 ”，“（null 文字（¥0）を含む）最後の 1 個以上の認識されない文字列 ” に分割します。その後，対象となる列を整数へ変換し，その結果を返します。

(1) 引数 *base* は，0，または 2 ～ 36 を指定します。

(a) *base* が 0 の場合
対象となる文字列の予期される形式は，オプショナルな + 符号，または - 符号，16 進数であることを示す “0x” を前に持つ整数の形式となります。

(b) *base* の値が 2 ～ 36 の場合
対象となる文字列の予期される形式は，オプショナルな + 符号，または - 符号を前に持ち，*base* によって基数が指定された整数を表す文字列，または数字列となります。“a”（または “A”）から “z”（または “Z”）までの英字は 10 から 35 までの値を示すものとみなされます。与えられた値が *base* よりも小さい英字しか使用できません。

(c) *base* の値が 16 の場合
“0x” が文字と数字の列の前（符号が存在する場合は符号の後ろ）に置かれます（省略可能）。

(2) 対象となる列は，空白類以外の最初の文字で始まり，予期される形式を持つ入力文字列の先頭部分の最長の部分列として定義されます。

(a) 入力の文字列が空である場合やすべて空白類で構成されている場合，または空白類でない最初の文字が符号でも許容されうる文字でも数字でもない場合，対象となる列は空となります。

(b) 対象となる列が予期される形式を持ち，かつ，*base* の値が 0 の場合，入力文字列から基数を判断します。0x が先行する文字列は，16 進数数値とみなされ，先行 0 が付いていて x が付いていない文字列は 8 進数としてみなされます。他の文字列はすべて 10 進数としてみなされます。

(c) *base* が 2 から 36 までの間の値の場合，上述のように，これを変換用基数として使用します。

(d) 対象となる列が - 符号で始まる場合，変換結果の値の符号は反転されます。

(3) 最初の文字列を指すポインタ

(a) *ptr* が null ポインタでない場合，*ptr* の指すオブジェクトの中に格納されます。

(b) 対象となる列が空である場合，または予期された形式を持たない場合，変換は行われません。*str* の値は，*ptr* が null ポインタでない場合，*ptr* の指すオブジェクトに格納されます。

備考　本関数は，リエントラントではありません。

[使用例]

```
#include    <stdlib.h>
void func(long ret) {
    char    *p;
    ret = strtol("10", &p, 0);      /*ret に 10 を返す */
    ret = strtol("0x10", &p, 0);    /*ret に 16 を返す */
    ret = strtol("10x", &p, 2);     /*ret に 2 を返し，p の領域には 'x' へのポインタを格納 */
    ret = strtol("2ax3", &p, 16);   /*ret に 42 を返し，p の領域には 'x' へのポインタを格納 */
      :
}
```

strtoul

文字列を整数（unsigned long 型）へ変換し，最終文字列へのポインタを格納します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
unsigned long   strtoul(const char *str, char **ptr, int base);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる場合，ULONG_MAX を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

返却値の型が unsigned long 型になること以外，strtol と同じです。

strtoll

文字列を整数（long long 型）へ変換し，最終文字列へのポインタを格納します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
long long   strtoll(const char *str, char **ptr, int base);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる（変換された値が大きすぎる）場合，LLONG_MAX，または LLONG_MIN を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

返却値の型が lng long 型になること以外，strtol と同じです。

[注意事項]

-Xansi オプション指定時の動作結果は不定です。

strtoull

文字列を整数（unsigned long long 型）へ変換し，最終文字列へのポインタを格納します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
unsigned long long  strtoull(const char *str, char **ptr, int base);
```

[戻り値]

部分文字列が変換できた場合，変換された値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる場合，ULLONG_MAX を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

返却値の型が unsigned long long 型になること以外，strtol と同じです。

[注意事項]

-Xansi オプション指定時の動作結果は不定です。

atoff

文字列を浮動小数点数（float 型）への変換を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
float   atoff(const char *str);
```

[戻り値]

部分文字列が変換できた場合，その値を返します。変換できなかった場合，0 を返します。オーバーフローが生じる（値が表現可能な値の範囲にない）場合，HUGE_VAL，または -HUGE_VAL を返し，ERANGE をグローバル変数 errno にセットします。アンダフローが生じる場合，0 を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

str の指す文字列の最初の部分を float 型の表現に変換します。本関数は，“strtodf (*str*, NULL)” と同じです。

atof

文字列を浮動小数点数（double 型）への変換を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
double  atof(const char *str);
```

[戻り値]

部分文字列が変換できた場合，その値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる（値が表現可能な値の範囲にない）場合，HUGE_VAL，または -HUGE_VAL を返し，ERANGE をグローバル変数 errno にセットします。アンダフローが生じる場合，0 を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

str の指す文字列の最初の部分を double 型の表現に変換します。本関数は，“strtod (*str*, NULL)” と同じです。

strtodf

文字列を浮動小数点数（float 型）への変換（最終文字列へのポインタ格納）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
float strtodf(const char **str*, char ***ptr*);

[戻り値]

部分文字列が変換できた場合，その値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる（値が表現可能な値の範囲にない）場合，HUGE_VAL，または -HUGE_VAL を返し，ERANGE をグローバル変数 errno にセットします。アンダフローが生じる場合，0 を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

str の指す文字列の最初の部分を float 型の表現に変換します。変換される部分文字列は，次の形式の，空白でない通常の文字から始まる，*str* の最長先頭部分文字列です。

[+ | -] digits [.] [digits] [(e | E) [+ | -] digits]

str が空か，または空白文字だけから成り立っている場合，および最初の通常文字が “+，“-”，“.”，または数字以外の場合，部分文字列には文字が含まれていません。部分文字列が空の場合，変換は行われず，str の値が *ptr* の指す領域に格納されます。空でない場合，部分文字列は変換され，最終文字列（少なくとも str の終端を示す null 文字（¥0）を含む）へのポインタが *ptr* の指す領域に格納されます。

備考　　本関数は，リエントラントではありません。

[使用例]

```
#include    <stdlib.h>
#include    <stdio.h>
void func(float ret) {
    char    *p, *str, s[30];
    str = "+5.32a4e";
    ret = strtodf(str, &p);         /*ret には 5.320000 を返し，p の領域には ‘a’ へのポインタを
*/
                                    /* 格納 */
    sprintf(s, "%lf\t%c", ret, *p); /*s の指す配列に “5.320000 a” を格納 */
}
```

strtod

文字列を浮動小数点数（double 型）への変換（最終文字列へのポインタ格納）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
double  strtod(const char *str, char **ptr);
```

[戻り値]

部分文字列が変換できた場合，その値を返します。変換できなかった場合，0 を返します。
オーバーフローが生じる（値が表現可能な値の範囲にない）場合，HUGE_VAL，または -HUGE_VAL を返し，ERANGE をグローバル変数 errno にセットします。アンダフローが生じる場合，0 を返し，マクロ ERANGE をグローバル変数 errno にセットします。

[詳細説明]

str の指す文字列の最初の部分を double 型の表現に変換します。変換される部分文字列は，次の形式の，空白でない通常の文字から始まる，*str* の最長先頭部分文字列です。

[+ | -] digits [.] [digits] [(e | E) [+ | -] digits]

str が空か，または空白文字だけから成り立っている場合，および最初の通常文字が “+，“-”，“.”，または数字以外の場合，部分文字列には文字が含まれていません。部分文字列が空の場合，変換は行われず，str の値が *ptr* の指す領域に格納されます。空でない場合，部分文字列は変換され，最終文字列（少なくとも str の終端を示す null 文字（¥0）を含む）へのポインタが *ptr* の指す領域に格納されます。

備考　　本関数は，リエントラントではありません。

calloc

メモリ割り当て（ゼロ初期化付き）を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
void *calloc(size_t *nmemb*, size_t *size*);

[戻り値]

領域の割り付けに成功した場合，その領域へのポインタを返します。割り付けができなかった場合，null ポインタを返します。

[詳細説明]

大きさが *size* の，要素数 *nmemb* 個の配列領域を割り付けます。割り付けられた領域は 0 で初期化されます。

[注意事項]

記憶域管理の関数は，ヒープ・メモリ領域から必要に応じて自動的にメモリ領域を確保します。
また，デフォルトのサイズは 0x1000 バイトなので，変更する場合は，ヒープ・メモリ領域を確保する必要があります。領域の確保は，アプリケーションの最初で行ってください。

【ヒープ・メモリ設定例】

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x2000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考 1. 変数 “_REL_sysheap” は，ヒープ・メモリの先頭アドレスを指します。この値は，4 の倍数にしてください。

備考 2. 変数 “_REL_sizeof_sysheap” に，必要なヒープ・メモリのサイズ（バイト）を設定してください。

[使用例]

```
#include    <stdlib.h>
typedef struct {
    double  d[3];
    int     i[2];
} s_data;
int func(void) {
    s_data   *buf;
    if((buf = calloc(40, sizeof(s_data))) == NULL)  /*s_data40 個のための領域を割り付け */
        return(1);
    /* 処理を記述 */
    free(buf);                                      /* 領域を開放 */
    return(0);
}
```

malloc

メモリ割り当て（ゼロ初期化なし）を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void    *malloc(size_t size);
```

[戻り値]

領域の割り付けに成功した場合，その領域へのポインタを返します。割り付けができなかった場合，null ポインタを返します。

[詳細説明]

大きさ *size* の領域を割り付けます。領域は初期化されません。

[注意事項]

記憶域管理の関数は，ヒープ・メモリ領域から必要に応じて自動的にメモリ領域を確保します。
また，デフォルトのサイズは 0x1000 バイトなので，変更する場合は，ヒープ・メモリ領域を確保する必要があります。領域の確保は，アプリケーションの最初で行ってください。

【ヒープ・メモリ設定例】

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x2000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考 1. 変数 “_REL_sysheap” は，ヒープ・メモリの先頭アドレスを指します。この値は，4 の倍数にしてください。

備考 2. 変数 “_REL_sizeof_sysheap” に，必要なヒープ・メモリのサイズ（バイト）を設定してください。

realloc

メモリの再割り当てを行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
void *realloc(void **ptr*, size_t *size*);

[戻り値]

領域の割り付けに成功した場合，その領域へのポインタを返します。割り付けができなかった場合，null ポインタを返します。

[詳細説明]

ptr が指す領域の大きさを，*size* の大きさに変更します。以前の大きさと，*size* の小さい方までの領域の内容は変わりません。領域を拡張する場合の，以前の大きさ以降の領域内容は初期化されません。*ptr* が null ポインタのときは，“malloc (*size*)” と同じ動作をします。それ以外の場合，*ptr* には，calloc，malloc，および本関数で獲得した領域を指定しなければなりません。

[注意事項]

記憶域管理の関数は，ヒープ・メモリ領域から必要に応じて自動的にメモリ領域を確保します。
また，デフォルトのサイズは 0x1000 バイトなので，変更する場合は，ヒープ・メモリ領域を確保する必要があります。領域の確保は，アプリケーションの最初で行ってください。

【ヒープ・メモリ設定例】

```
#include    <stddef.h>
#define     SIZEOF_HEAP 0x2000
int     _REL_sysheap[SIZEOF_HEAP >> 2];
size_t  _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考 1. 変数 “_REL_sysheap” は，ヒープ・メモリの先頭アドレスを指します。この値は，4 の倍数にしてください。

備考 2. 変数 “_REL_sizeof_sysheap” に，必要なヒープ・メモリのサイズ（バイト）を設定してください。

free

メモリ開放を行います。

[所属]

標準ライブラリ

[指定形式]

#include <stdlib.h>
void free(void *ptr);

[詳細説明]

ptr が指す領域を開放し，その後の割り付けに使用できるようにします。*ptr* には，calloc，malloc，および realloc で獲得した領域を指定しなければなりません。

[使用例]

```
#include    <stdlib.h>
typedef struct {
    double  d[3];
    int     i[2];
} s_data;
int func(void) {
    s_data   *buf;
    if((buf = calloc(40, sizeof(s_data))) == NULL)  /*s_data40 個のための領域を割り付け */
        return(1);

    free(buf);                                      /* 領域を開放 */
    return(0);
}
```

rand

疑似乱数列生成を行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
int rand(void);
```

[戻り値]

乱数を返します。

[詳細説明]

0 以上 RAND_MAX 以下の乱数を返します。

[使用例]

```
#include    <stdlib.h>
void func(void) {
    if((rand() & 0xF) < 4)
        func1();    /*25%の確率で func1 を実行 */
}
```

srand

疑似乱数列の種類を設定します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void    srand(unsigned int seed);
```

[詳細説明]

後続する rand の呼び出しで使用する新しい疑似乱数列の種として，*seed* を与えます。本関数を同じ *seed* の値で呼んだ場合，rand により得られる乱数は，同じ値が同じ順番で現れることになります。本関数を実行せずに rand を実行した場合，最初に “srand (1)” を実行した場合と同じ結果になります。

abort

プログラムを異常終了します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void   abort(void);
```

[詳細説明]

abort(void) は，プログラムを異常終了します。使用する場合，ユーザ・システムに合わせて abort 関数を作成する必要があります。

[使用例]

```
#include    <assert.h>
int func(void);
int main() {
    int ret;
    ret = func();
    if (ret == 0) {
        abort();                ← ret が 0 以外のとき，abort() が呼ばれる
    }
    return 0;
}
```

7.4.9 非局所分岐関数

非局所分岐関数として，以下のものがあります。

表 7.11 非局所分岐関数

関数／マクロ名	概要
longjmp	非局所分岐
setjmp	非局所分岐の分岐先をセット

longjmp

非局所分岐を行います。

[所属]

標準ライブラリ

[指定形式]

#include <setjmp.h>
void longjmp(jmp_buf *env*, int *val*);

[詳細説明]

setjmp で保存された *env* を使い，setjmp 直後へ非局所分岐します。*val* は，setjmp の返却値となります。

【setjmp.h における jmp_buf 型の定義】

```
typedef int     jmp_buf[14];
```

[注意事項]

本関数が退避／復帰するのはコンパイラで予約したレジスタのみです。
22 レジスタ・モード，または共通レジスタ・モードの関数内で setjmp を呼び出し，32 レジスタ・モードの関数内で r20 ～ r24 を破壊した後に本関数を呼び出した場合，r20 ～ r24 の値は回復しません。本関数を呼び出す前に，r20 ～ r24 を回復しておく必要があります。

[使用例]

```
#include    <setjmp.h>
#define ERR_XXX1    1
#define ERR_XXX2    2

jmp_buf jmp_env;

void main(void) {
    for(;;) {
        switch(setjmp(jmp_env)) {
            case    ERR_XXX1:
                /*error XXX1 の終結処理 */
                break;
            case    ERR_XXX2:
                /*error XXX2 の終結処理 */
                break;
            case    0:          /* 非局所分岐ではない */
            default:
                break;
        }
    }
}

void funcXXX(void) {
    longjmp(jmp_env, ERR_XXX1); /*error XXX1 の発生により非局所分岐 */
    longjmp(jmp_env, ERR_XXX2); /*error XXX2 の発生により非局所分岐 */
}
```

setjmp

非局所分岐の分岐先をセットします。

[所属]

標準ライブラリ

[指定形式]

#include <setjmp.h>
int setjmp(jmp_buf *env*);

[戻り値]

setjmp からの戻りの場合 0 を返します。longjmp によって非局所分岐の場合，longjmp の第 2 引数 val を返します。ただし，val が 0 の場合，1 を返します。

[詳細説明]

非局所分岐のための戻り先を *env* にセットします。*env* には，本関数が実行された時点の環境が保存されます。

【setjmp.h における jmp_buf 型の定義】

```
typedef int     jmp_buf[14];
```

[注意事項]

本関数が保存，回復するのはコンパイラで予約したレジスタのみです。
22 レジスタ・モード，または共通レジスタ・モードの関数内で本関数を呼び出し，32 レジスタ・モードの関数内で r20 ～ r24 を破壊した後に longjmp を呼び出した場合，r20 ～ r24 の値は回復しません。longjmp を呼び出す前に，r20 ～ r24 を回復しておく必要があります。

7.4.10 数学関数

数学関数として，以下のものがあります。

表 7.12 数学関数

関数／マクロ名	概要
expf	指数関数
exp	指数関数
logf	対数関数（自然対数）
log	対数関数（自然対数）
log10f	対数関数（底 = 10）
log10	対数関数（底 = 10）
powf	べき乗関数
pow	べき乗関数
sqrtf	平方根関数
sqrt	平方根関数
ceilf	ceiling 関数
ceil	ceiling 関数
fabsf	絶対値関数
fabs	絶対値関数
floorf	floor 関数
floor	floor 関数
fmodf	剰余関数
fmod	剰余関数
frexpf	浮動小数点数を仮数部とべき乗に分割
frexp	浮動小数点数を仮数部とべき乗に分割
ldexpf	浮動小数点数をべき乗に変換
ldexp	浮動小数点数をべき乗に変換
modff	浮動小数点数を整数部と小数部に分割
modf	浮動小数点数を整数部と小数部に分割
cosf	余弦
cos	余弦
sinf	正弦
sin	正弦
tanf	正接
tan	正接
acosf	逆余弦
acos	逆余弦

関数／マクロ名	概要
asinf	逆正弦
asin	逆正弦
atanf	逆正接
atan	逆正接
atan2f	逆正接（y / x）
atan2	逆正接（y / x）
coshf	双曲線余弦
cosh	双曲線余弦
sinhf	双曲線正弦
sinh	双曲線正弦
tanhf	双曲線正接
tanh	双曲線正接

expf

指数関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   expf(float x);
```

[戻り値]

e の *x* 乗を返します。
アンダフローが生じた場合（*x* が結果を表せない大きさの負の数の場合），非正規化数を返し，グローバル変数 errno にマクロ ERANGE をセットします。オーバーフローが生じた場合（*x* が大きすぎる数の場合），HUGE_VAL（表現可能な最大の double 型数値）を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

e の *x* 乗を求めます（e は自然対数の底で，約 2.71828 です）。

exp

指数関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  exp(double x);
```

[戻り値]

e の *x* 乗を返します。
アンダフローが生じた場合（*x* が結果を表せない大きさの負の数の場合），非正規化数を返し，グローバル変数 errno にマクロ ERANGE をセットします。オーバーフローが生じた場合（*x* が大きすぎる数の場合），HUGE_VAL（表現可能な最大の double 型数値）を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

e の *x* 乗を求めます（e は自然対数の底で，約 2.71828 です）。

logf

対数関数（自然対数）です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   logf(float x);
```

[戻り値]

x の自然対数を返します。
x が負の場合非数を返し，グローバル変数 errno にマクロ EDOM をセットします。*x* が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

x の自然対数，つまり，底を e としてその対数を求めます。

log

対数関数（自然対数）です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  log(double x);
```

[戻り値]

x の自然対数を返します。
x が負の場合非数を返し，グローバル変数 errno にマクロ EDOM をセットします。*x* が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

x の自然対数，つまり，底を e としてその対数を求めます。

log10f

対数関数（底 = 10）です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   log10f(float x);
```

[戻り値]

10 を底とする *x* の対数を返します。
x が負の場合非数を返し，グローバル変数 errno にマクロ EDOM をセットします。x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

10 を底とする *x* の対数を求めます。これは，“log (*x*) / log (10)” によって実現されています。

log10

対数関数（底 = 10）です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double   log10(double x);
```

[戻り値]

10 を底とする *x* の対数を返します。
x が負の場合非数を返し，グローバル変数 errno にマクロ EDOM をセットします。x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE をセットします。

[詳細説明]

10 を底とする *x* の対数を求めます。これは，“log (*x*) / log (10)” によって実現されています。

powf

べき乗関数です。

[所属]

数学ライブラリ

[指定形式]

#include <mathf.h>
float powf(float *x*, float *y*);

[戻り値]

x の *y* 乗を返します。
x < 0 かつ *y* が奇整数の場合にのみ負の解を返します。*x* < 0 で *y* が非整数の場合，または *x* = *y* = 0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。*x* = 0 かつ *y* < 0 の場合，またはオーバーフローが発生した場合，±HUGE_VAL を返し，errno にマクロ ERANGE をセットします。解が 0 へ向かって消滅した場合，0 を返し，errno に ERANGE をセットします。解が非正規化数の場合，errno に ERANGE をセットします。

[詳細説明]

x の *y* 乗を求めます。

[使用例]

```
#include    <mathf.h>
float func(void) {
    float   ret, x, y;
    ret = powf(x, y);    /*x を y 乗した値を ret に返す */
      :
    return(ret);
}
```

pow

べき乗関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  pow(double x, double y);
```

[戻り値]

x の *y* 乗を返します。
x < 0 かつ *y* が奇整数の場合にのみ負の解を返します。*x* < 0 で *y* が非整数の場合，または *x* = *y* = 0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。*x* = 0 かつ *y* < 0 の場合，またはオーバーフローが発生した場合，±HUGE_VAL を返し，errno にマクロ ERANGE をセットします。解が 0 へ向かって消滅した場合，0 を返し，errno に ERANGE をセットします。解が非正規化数の場合，errno に ERANGE をセットします。

[詳細説明]

x の *y* 乗を求めます。

sqrtf

平方根関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   sqrtf(float x);
```

[戻り値]

x の正の平方根を返します。
x が実数で負の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。

[詳細説明]

x の正の平方根を求めます。

sqrt

平方根関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  sqrt(double x);
```

[戻り値]

x の正の平方根を返します。
x が実数で負の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。

[詳細説明]

x の正の平方根を求めます。

ceilf

ceiling 関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   ceilf(float x);
```

[戻り値]

x 以上の最小の整数値を返します。

[詳細説明]

x 以上の最小の整数値を求めます。

ceil

ceiling 関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  ceil(double x);
```

[戻り値]

x 以上の最小の整数値を返します。

[詳細説明]

x 以上の最小の整数値を求めます。

fabsf

絶対値関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   fabsf(float x);
```

[戻り値]

x の絶対値（大きさ）を返します。

[詳細説明]

x のビット表現を直接操作して，*x* の絶対値（大きさ）を求めます。

fabs

絶対値関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  fabs(double x);
```

[戻り値]

x の絶対値（大きさ）を返します。

[詳細説明]

x のビット表現を直接操作して，*x* の絶対値（大きさ）を求めます。

floorf

floor 関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   floorf(float x);
```

[戻り値]

x 以下の最大の整数値を返します。

[詳細説明]

x 以下の最大の整数値を求めます。

floor

floor 関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  floor(double x);
```

[戻り値]

x 以下の最大の整数値を返します。

[詳細説明]

x 以下の最大の整数値を求めます。

fmodf

剰余関数です。

[所属]

数学ライブラリ

[指定形式]

#include <mathf.h>
float fmodf(float *x*, float *y*);

[戻り値]

x を *y* で割った剰余である浮動小数点数値を返します。
y が ±∞の場合，*x* を返します。
x が ±∞の場合，または *y* が 0 の場合，非数を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x を *y* で割った剰余である浮動小数点数値を求めます。つまり，*y* が 0 でない場合に，その結果の符号が *x* と同じ符号で大きさが *y* よりも小さい最大整数 i に対し，値 “*x* - i * *y*” を求めます。

[使用例]

```
#include    <mathf.h>
void func(void) {
    float   ret, x, y;
    ret = fmodf(x, y);  /*x を y で割った剰余を ret に返す */
      :
}
```

fmod

剰余関数です。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  fmod(double x, double y);
```

[戻り値]

x を *y* で割った剰余である浮動小数点数値を返します。
y が ± ∞の場合，*x* を返します。
x が ± ∞の場合，または *y* が 0 の場合，非数を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x を *y* で割った剰余である浮動小数点数値を求めます。つまり，*y* が 0 でない場合に，その結果の符号が *x* と同じ符号で大きさが *y* よりも小さい最大整数 i に対し，値 “*x* - i * *y*” を求めます。

frexpf

浮動小数点数を仮数部とべき乗に分割します。

[所属]

数学ライブラリ

[指定形式]

#include <mathf.h>
float frexpf(float *val*, int **exp*);

[戻り値]

仮数部 m を返します。
val が 0 の場合，**exp* に 0 をセットし，0 を返します。
val が ± ∞の場合，0 を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

float 型の *val* を仮数部 m と 2 の p 乗で表します。結果の仮数部 m は，*val* が 0 でないかぎり，0.5 <= | x | < 1.0 となります。p は **exp* に格納されます。m，および p は，*val* = m * 2^p となるように計算されます。

[使用例]

```
#include    <mathf.h>
void func(void) {
    float   ret, x;
    int     exp;
    x = 5.28;
    ret = frexpf(x, &exp);  /* 結果の仮数部 0.66 が ret に返り，exp には 3 を格納 */
      :
}
```

frexp

浮動小数点数を仮数部とべき乗に分割します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  frexp(double val, int *exp);
```

[戻り値]

仮数部 m を返します。
val が 0 の場合，**exp* に 0 をセットし，0 を返します。
val が ± ∞の場合，0 を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

double 型の *val* を仮数部 m と 2 の p 乗で表します。結果の仮数部 m は，*val* が 0 でないかぎり，0.5 <= | x | < 1.0 となります。p は **exp* に格納されます。m，および p は，*val* = m * 2^p となるように計算されます。

ldexpf

浮動小数点数をべき乗に変換します。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   ldexpf(float val, int exp);
```

[戻り値]

val * 2 ^{exp} で求めた値を返します。
アンダフロー，またはオーバーフローが生じた場合，グローバル変数 errno にマクロ ERANGE がセットされます。アンダフローの場合，非正規化数を返します。オーバーフローの場合，HUGE_VAL を返します。

[詳細説明]

val * 2 ^{exp} を求めます。

ldexp

浮動小数点数をべき乗に変換します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  ldexp(double val, int exp);
```

[戻り値]

val * 2 ^{exp} で求めた値を返します。
アンダフロー，またはオーバーフローが生じた場合，グローバル変数 errno にマクロ ERANGE がセットされます。アンダフローの場合，非正規化数を返します。オーバーフローの場合，HUGE_VAL を返します。

[詳細説明]

val * 2 ^{exp} を求めます。

modff

浮動小数点数を整数部と小数部に分割します。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   modff(float val, float *ipart);
```

[戻り値]

小数部を返します。結果の符号は *val* の符号と同じです。

[詳細説明]

float 型の *val* を整数部と小数部とに分割し，整数部を **ipart* に格納します。丸めは行いません。整数部と小数部の和は，正確に *val* と一致するように保証されています。たとえば，“*realpart* = modff (*val*, &*intpart*)” であるとき，“*realpart* + *intpart*” は *val* と一致します。

modf

浮動小数点数を整数部と小数部に分割します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  modf(double val, double *ipart);
```

[戻り値]

小数部を返します。結果の符号は *val* の符号と同じです。

[詳細説明]

double 型の *val* を整数部と小数部とに分割し，整数部を **ipart* に格納します。丸めは行いません。整数部と小数部の和は，正確に *val* と一致するように保証されています。たとえば，“*realpart* = modf (*val*, &*intpart*)” であるとき，“*realpart* + *intpart*” は *val* と一致します。

cosf

余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   cosf(float x);
```

[戻り値]

x の余弦を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

x の余弦を求めます。角度はラジアン単位で指定します。

cos

余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  cos(double x);
```

[戻り値]

x の余弦を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

x の余弦を求めます。角度はラジアン単位で指定します。

sinf

正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   sinf(float x);
```

[戻り値]

x の正弦を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の正弦を求めます。角度はラジアン単位で指定します。

sin

正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  sin(double x);
```

[戻り値]

x の正弦を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の正弦を求めます。角度はラジアン単位で指定します。

tanf

正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   tanf(float x);
```

[戻り値]

x の正接を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の正接を求めます。角度はラジアン単位で指定します。

tan

正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  tan(double x);
```

[戻り値]

x の正接を返します。
± ∞を入力した場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の正接を求めます。角度はラジアン単位で指定します。

acosf

逆余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   acosf(float x);
```

[戻り値]

x の逆余弦（アークコサイン）を返します。返す値はラジアン単位で，0 から π までの範囲です。
x が -1 と 1 の間にない場合，非数を返します。また，グローバル変数 errno にマクロ EDOM をセットます。

[詳細説明]

x の逆余弦（アークコサイン）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

acos

逆余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  acos(double x);
```

[戻り値]

x の逆余弦（アークコサイン）を返します。返す値はラジアン単位で，0 から π までの範囲です。
x が -1 と 1 の間にない場合，非数を返します。また，グローバル変数 errno にマクロ EDOM をセットます。

[詳細説明]

x の逆余弦（アークコサイン）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

asinf

逆正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   asinf(float x);
```

[戻り値]

x の逆正弦（アークサイン）を返します。返す値はラジアン単位で，-π / 2 から π / 2 までの範囲です。
x が -1 と 1 の間にない場合，非数を返します。また，グローバル変数 errno にマクロ EDOM をセットします。

[詳細説明]

x の逆正弦（アークサイン）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

asin

逆正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  asin(double x);
```

[戻り値]

x の逆正弦（アークサイン）を返します。返す値はラジアン単位で，-π / 2 から π / 2 までの範囲です。
x が -1 と 1 の間にない場合，非数を返します。また，グローバル変数 errno にマクロ EDOM をセットします。

[詳細説明]

x の逆正弦（アークサイン）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

atanf

逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

#include <mathf.h>
float atanf(float *x*);

[戻り値]

x の逆正接（アークタンジェント）を返します。返す値はラジアン単位で，-π / 2 から π / 2 の範囲です。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の逆正接（アークタンジェント）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

[使用例]

```
#include    <mathf.h>
float func(float x) {
    float   ret;
    ret = atanf(x); /*x の逆正接の値を ret に返す */
      :
    return(ret);
}
```

atan

逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  atan(double x);
```

[戻り値]

x の逆正接（アークタンジェント）を返します。返す値はラジアン単位で，-π / 2 から π / 2 の範囲です。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の逆正接（アークタンジェント）を求めます。*x* は，-1 ≦ *x* ≦ 1 で指定します。

atan2f

逆正接（y / x）を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   atan2f(float y, float x);
```

[戻り値]

y / *x* の逆正接（アークタンジェント）を返します。返す値はラジアン単位で，-π から π までの範囲です。
x と *y* がともに 0.0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。解が 0 へ向かって消滅した場合，±0 を返し，グローバル変数 errno にマクロ ERANGE をセットします。解が非正規化数の場合，グローバル変数 errno に ERANGE をセットします。

[詳細説明]

y / *x* の逆正接（アークタンジェント）を求めます。本関数は，π / 2，または -π / 2 付近（*x* が 0 に近い場合）の角度の場合も正しい結果を求めます。

atan2

逆正接（y / x）を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  atan2(double y, double x);
```

[戻り値]

y / *x* の逆正接（アークタンジェント）を返します。返す値はラジアン単位で，-π から π までの範囲です。
x と *y* がともに 0.0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM をセットします。解が 0 へ向かって消滅した場合，±0 を返し，グローバル変数 errno にマクロ ERANGE をセットします。解が非正規化数の場合，グローバル変数 errno に ERANGE をセットします。

[詳細説明]

y / *x* の逆正接（アークタンジェント）を求めます。本関数は，π / 2，または -π / 2 付近（*x* が 0 に近い場合）の角度の場合も正しい結果を求めます。

coshf

双曲線余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   coshf(float x);
```

[戻り値]

x の双曲線余弦を返します。
オーバーフローが生じた場合，HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE をセットします。

[詳細説明]

x の双曲線余弦を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

$(e^{x} + e^{-x}) / 2$

[使用例]

```
#include    <mathf.h>
float func(float x){
    float   ret;
    ret = coshf(x);      /*x の双曲線余弦の値を ret に返す */
      :
    return(ret);
}
```

cosh

双曲線余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  cosh(double x);
```

[戻り値]

x の双曲線余弦を返します。
オーバーフローが生じた場合，HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE をセットします。

[詳細説明]

x の双曲線余弦を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

$(e^{x} + e^{-x}) / 2$

sinhf

双曲線正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   sinhf(float x);
```

[戻り値]

x の双曲線正弦を返します。
オーバーフローが生じた場合，HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE をセットします。

[詳細説明]

x の双曲線正弦を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

$(e^{x} - e^{-x}) / 2$

sinh

双曲線正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  sinh(double x);
```

[戻り値]

x の双曲線正弦を返します。
オーバーフローが生じた場合，HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE をセットします。

[詳細説明]

x の双曲線正弦を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

$(e^{x} - e^{-x}) / 2$

tanhf

双曲線正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <mathf.h>
float   tanhf(float x);
```

[戻り値]

x の双曲線正接を返します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の双曲線正接を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

sinh (*x*) / cosh (*x*)

tanh

双曲線正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  tanh(double x);
```

[戻り値]

x の双曲線正接を返します。
解が非正規化数の場合，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の双曲線正接を求めます。角度はラジアン単位で指定します。定義式は次のとおりです。

sinh (*x*) / cosh (*x*)

7.4.11 RAM セクション領域初期化関数

RAM セクション領域初期化関数として，以下のものがあります。

表 7.13 RAM セクション領域初期化関数

関数／マクロ名	概要
_INITSCT_RH	RAM 領域セクションの初期値コピーとゼロ・クリア

_INITSCT_RH

RAM 領域セクションの初期値コピーとゼロ・クリアを行います。

[所属]

標準ライブラリ

[指定形式]

```
#include <_h_c_lib.h>
void _INITSCT_RH(void * datatbl_start, void * datatbl_end, void * bsstbl_start, void * bsstbl_end)
```

[引数／戻り値]

引数	戻り値
datatbl_start ： data 属性セクション初期化テーブルの先頭アドレス datatbl_end ： data 属性セクション初期化テーブルの末尾アドレス bsstbl_start： bss 属性セクション初期化テーブルの先頭アドレス bsstbl_end ： bss 属性セクション初期化テーブルの末尾アドレス	なし

[詳細説明]

RAM 領域セクションのうち，data 属性セクションの初期値を ROM 領域からコピーし，bss 属性セクションのゼロ・クリアを行います。
第 1，第 2 引数は，data 属性セクションの初期化テーブルの先頭，末尾アドレスを渡します。
第 3，第 4 引数は，bss 属性セクションの初期化テーブルの先頭，末尾アドレスを渡します。

第 1 引数＞＝第 2 引数の場合，data 属性セクションの初期化を行いません。
第 3 引数＞＝第 4 引数の場合，bss 属性セクションのゼロ・クリアを行いません。

[使用例]

```
struct {
        void *rom_s;     //data 属性セクションの ROM 上の先頭アドレス
        void *rom_e;     //data 属性セクションの ROM 上の末尾アドレス
        void *ram_s;     //data 属性セクションの RAM 上の先頭アドレス
} _C_DSEC[M];

struct {
        void *bss_s;     //bss 属性セクションの RAM 上の先頭アドレス
        void *bss_e;     //bss 属性セクションの RAM 上の末尾アドレス
} _C_BSEC[N];

_INITSCT_RH(_C_DSEC, _C_DSEC + M, _C_BSEC, _C_BSEC + N);
```

備考　.bss セクションの開始アドレスが 0x100，サイズが 0x50 バイトであるとき，0 クリアされるメモリは 0x100，0x101，...，0x14e，0x14f 番地ですが，初期化テーブルには 0x100, 0x150 と指定します。

7.4.12 周辺装置の初期化関数

周辺装置の初期化関数として，以下のものがあります。

表 7.14 周辺装置の初期化関数

関数／マクロ名	概要
hdwinit	CPU リセット直後の周辺装置の初期化処理

hdwinit

CPU リセット直後に周辺装置の初期化処理を行います。

[所属]

標準ライブラリ

[指定形式]

```
void    hdwinit(void);
```

[詳細説明]

周辺装置の初期化関数は，CPU リセット直後に周辺装置の初期化処理を行う関数です。
スタートアップ・ルーチン内から呼び出されます。
ライブラリに含まれる関数は，実動作は何もしないダミー・ルーチンですので，システムに合わせて，記述してください。

7.4.13 演算用ランタイム関数

演算用ランタイム関数として，以下のものがあります。

表 7.15 演算用ランタイム関数

分類	関数名	概要
float 型演算関数	_COM_fadd	単精度浮動小数点の加算
	_COM_fsub	単精度浮動小数点の減算
	_COM_fmul	単精度浮動小数点の乗算
	_COM_fdiv	単精度浮動小数点の除算
double 型演算関数	_COM_dadd	倍精度浮動小数点型の加算
	_COM_dsub	倍精度浮動小数点型の減算
	_COM_dmul	倍精度浮動小数点型の乗算
	_COM_ddiv	倍精度浮動小数点型の除算
long long 型演算関数	_COM_mul64	64 ビット整数の乗算
	_COM_div64	符号付き 64 ビット整数の除算
	_COM_udiv64	符号なし 64 ビット整数の除算
	_COM_rem64	符号付き 64 ビット整数の剰余算
	_COM_urem64	符号なし 64 ビット整数の剰余算
	_COM_shll_64_32	64 ビット整数の論理左シフト演算
	_COM_shrl_64_32	64 ビット整数の論理右シフト演算
	_COM_shra_64_32	64 ビット整数の算術右シフト演算
	_COM_neg64	符号反転

分類	関数名	概要
型変換関数	_COM_itof	32 ビット整数から単精度浮動小数点への変換
	_COM_itod	32 ビット整数から倍精度浮動小数点への変換
	_COM_utof	符号なし 32 ビット整数から単精度浮動小数点への変換
	_COM_utod	符号なし 32 ビット整数から倍精度浮動小数点への変換
	_COM_i64tof	64 ビット整数から単精度浮動小数点への変換
	_COM_i64tod	64 ビット整数から倍精度浮動小数点への変換
	_COM_u64tof	符号なし 64 ビット整数から単精度浮動小数点への変換
	_COM_u64tod	符号なし 64 ビット整数から倍精度浮動小数点への変換
	_COM_ftoi	単精度浮動小数点数から 32 ビット整数への変換
	_COM_dtoi	倍精度浮動小数点数から 32 ビット整数への変換
	_COM_ftou	単精度浮動小数点数から符号なし 32 ビット整数への変換
	_COM_dtou	倍精度浮動小数点数から符号なし 32 ビット整数への変換
	_COM_ftoi64	単精度浮動小数点数から 64 ビット整数への変換
	_COM_dtoi64	倍精度浮動小数点数から 64 ビット整数への変換
	_COM_ftou64	単精度浮動小数点数から符号なし 64 ビット整数への変換
	_COM_dtou64	倍精度浮動小数点数から符号なし 64 ビット整数への変換
	_COM_ftod	単精度浮動小数点数から倍精度浮動小数点への変換
	_COM_dtof	倍精度浮動小数点数から単精度浮動小数点への変換
浮動小数点比較演算関数	_COM_fgt	比較
	_COM_fge	比較
	_COM_feq	比較
	_COM_fne	比較
	_COM_flt	比較
	_COM_fle	比較
	_COM_funord	順序付け不可
	_COM_dgt	比較
	_COM_dge	比較
	_COM_deq	比較
	_COM_dne	比較
	_COM_dlt	比較
	_COM_dle	比較
	_COM_dunord	順序付け不可
ソフトウエア・トレース関数	_REL_tracestring	文字列出力

8. スタートアップ

この章では，スタートアップについて説明します。

8.1 概　　要

スタートアップは，C 言語により記述されたユーザ・アプリケーションをシステムへ組み込むためのセクションの初期化処理，main 関数の起動などを行うための処理です。
ここではコアを 1 つのみ使用する「シングルコア用のプログラム」，コアを複数使用する「マルチコア用のプログラム」の 2 種類を想定して説明します。

- シングルコア用のプログラムは，アプリケーション・プロジェクトを 1 つのみ用意します。
- マルチコア用のプログラムは，ブート・ローダ・プロジェクトを 1 つとアプリケーション・プロジェクトを必要な数だけ用意します。

これらのプログラムを動作させるための基本的なスタートアップ・ルーチンの構成例を R7F701Z07，R7F7010352AFP を例に示します。

8.2 スタートアップ・ルーチン

スタートアップ・ルーチンとは，マイクロコントローラをリセットしたあと，main 関数を実行する前に，実行するルーチンを言います。基本的にはシステムをリセットしたあとの初期化を行います。
ここでは，下記について説明します。

- 例外ベクタ・テーブル
- ブート・ローダ・プロジェクト用スタートアップ・ルーチン
- アプリケーション・プロジェクト用スタートアップ・ルーチン
- アプリケーション・プロジェクトからブート・ローダ・プロジェクトへの情報の受け渡し

8.2.1 例外ベクタ・テーブル

例外ベクタ・テーブルにはマイクロコントローラのリセット時やその他の例外発生時の分岐先を設定します。
サンプルでは vecttbl.asm に記述されており，以下のプロジェクトから参照します。
　シングルコア用のプログラムの場合 : アプリケーション・プロジェクト
　マルチコア用のプログラムの場合 : ブート・ローダ・プロジェクト

vecttbl.asm は次の要素で構成されます。

- RESET ベクタ
- 割り込みハンドラ・テーブル
- 例外ハンドラ・ルーチン

(1) RESET ベクタ
マイクロコントローラ起動時，およびリセット時にプログラムカウンタが分岐してくるアドレスに，PE（プロセッサ・エレメント）共通のエントリ・ポイント・ルーチンへの分岐命令を配置します。
サンプルでは RESET セクションに配置しています。セクション名は任意に変更可能ですが，最適化リンカの -start オプションと連動して変更する必要があります。

```
-start=RESET/01000000
```

注意　RESET ベクタのアドレスはデバイスのユーザーズ・マニュアルを参照してください。

EBASE レジスタは RBASE レジスタと同じ値を保持し，RESET を含む例外ベクタ・テーブルを共有するものとして扱います。RESET セクション内に標準仕様（直接ベクタ方式）の各例外ハンドラ・アドレスへの分岐命令を配置します。
PSW.EBV レジスタの状態によらず RBASE = EBASE として動作させるにはスタートアップ内に以下の記述が必要です。

```
stsr    2, r10, 1               ; get RBASE
ldsr    r10, 3, 1               ; set EBASE
```

備考　デバイスの初期状態は PSW.EBV = 0 であり，EBASE を使用しないので必須ではありません。

アドレスを保持する RBASE，EBASE レジスタが 512 バイト単位でアドレスを保持するため，RESET セクションの先頭は 512 バイトに整列します。

注意　RESET セクションの配置アドレス，ベクタ数はデバイスのユーザーズ・マニュアルを参照してください。

(2) 割り込みハンドラ・テーブル
拡張仕様（テーブル参照方式）の例外ハンドラ・アドレスを使用する場合，使用する例外ハンドラ・ルーチンのアドレスを本テーブルの対応する要素位置に配置します。
サンプルでは EIINTTBL セクションに配置しています。セクション名は任意に変更可能ですが，スタートアップ内の INTBP レジスタ設定処理と連動して変更する必要があります。

```
mov     #__sEIINTTBL, r10
ldsr    r10, 4, 1               ; set INTBP
```

テーブル・アドレスを保持する INTBP レジスタが 512 バイト単位でアドレスを保持するため，EIINTTBL セクションの先頭は 512 バイトに整列します。

注意　テーブルの最大要素数はデバイスのユーザーズ・マニュアルを参照してください。

(3) 例外ハンドラ・ルーチン
FE，EI 各レベルの例外ハンドラ・ルーチンのサンプルです。何もせず自分自身への分岐を繰り返します。
通常は，C ソース記述で #pragma interrupt を用いて用意します。

8.2.2 ブート・ローダ・プロジェクト用スタートアップ・ルーチン

ブート・ローダ・プロジェクト用スタートアップ・ルーチンはマルチコア用のプログラムで使用します。
boot.asm と vecttbl.asm（例外ベクタ・テーブル）の 2 つのファイルで構成されます。

boot.asm は次の要素で構成されます。

- マルチコア用のプログラムに必要な処理
  - PE 共通のエントリ・ルーチン
  - __exit ルーチン
  - PE1 のエントリ・ルーチン
  - PE3 のエントリ・ルーチン
- 必要に応じて用意する処理
  - hdwinit_PE1 ルーチン
  - hdwinit_PE3 ルーチン
  - zeroclr4 ルーチン
  - init_eiint ルーチン

(1) PE 共通のエントリ・ルーチン
各 PE が共通に実行する処理です。自分の PEID（プロセッサ・エレメント番号）値を読み出し，その値から PEID ごとに用意されたエントリ・ルーチンへ分岐します。アプリケーションを用意しない PE の場合は，直下の __exit ルーチンへ合流します。

(2) __exit ルーチン
使用しない PE を待機させておくための，自分自身への分岐を繰り返すのみのルーチンです。

(3) PE1 のエントリ・ルーチン
以下の手順で，PE1 用アプリケーション・プロジェクトへの分岐までを行います。

(a) PE1 用のハードウエア初期化ルーチン hdwinit_PE1 ( ) を呼び出します。

(b) EI レベル例外設定ルーチン init_eiint ( ) を呼び出します。サンプルでは無効化してあり，有効にするにはファイル冒頭の USE_TABLE_REFERENCE_METHOD マクロを有効にしてください。

備考 後述の hdwinit_PE1 ルーチン，init_eiint ルーチン の説明を参照し，処理が不要な場合には，呼び出す必要はありません。

(c) FEPC レジスタを経由して PC の設定を行います。
アプリケーション・プロジェクトで作成したアプリケーション情報テーブルから，アプリケーション・プロジェクトのエントリ・ルーチンのアドレスを読み出して設定します。

(d) アプリケーションのエントリ・ルーチンへ分岐します。feret 命令の実行によって，上記 FEPC レジスタ値を PC に反映し，アプリケーションのエントリ・ルーチンへの分岐を行います。

(4) PE3 のエントリ・ルーチン
基本的に PE1 のエントリ・ルーチンと同一の構成ですが，サンプルではアプリケーションへの分岐ではなく __exit ルーチンへの分岐で処理を終えています。

(5) hdwinit_PE1 ルーチン
以下の手順で，ハードウエアで 1 回行う初期化と PE1 固有の初期化処理を行います。

(a) ECC 機能向けに Global RAM と Local RAM（PE1）を初期化します。サンプルでは R7F701Z07 を対象としたアドレス範囲を指定し，Local RAM（self）経由で初期化しています。

注意 初期化対象の RAM アドレスはデバイスのユーザーズ・マニュアルを参照してください。

(b) PE3 の初期化処理が終わるまで待ち合わせを行います。手順を下記に示します。
待ち合わせ処理のフラグとして相互排他変数 MEV（mutual exclusion variable）を使用します。boot.asm では，MEV のアドレスを値として持つマクロ “MEV_ADDR” により MEV への操作を行っています。

- 相互排他変数 MEV を用意し，0 で初期化します。
- PE1 自身の処理完了を示す MEV のビット 0 に 1 を立てます。
- PE3 の処理完了を示す MEV のビット 1 の値を調べます。
- MEV のビット 1 の値が 1 であれば，PE1，PE3 とも初期化処理が完了したことになるので待ち合わせ処理を抜けます。0 であれば，PE3 がまだ初期化処理中なので，2 つ目の処理に戻り待ち合わせを続けます。

備考 1. PE1 と PE3 で処理を同期する必要がない場合はこの処理は不要です。

備考 2. MEV を 0 で初期化した後にビット 0 に 1 を立てる処理を続けるのは，ECC エラー対策です。MEV を配置する Global RAM は PE1 自身が初期化しているので実際は初期化を行う必要はありませんが，PE3 との対称性のために行なっています。

(6) hdwinit_PE3 ルーチン
以下の手順で，PE3 固有の初期化処理を行います。

(a) ECC 機能向けに Local RAM（PE3）を初期化します。サンプルでは R7F701Z07 を対象としたアドレス範囲を指定し，Local RAM（self）経由で初期化しています。

注意 初期化対象の RAM アドレスはデバイスのユーザーズ・マニュアルを参照してください。

(b) PE1 の初期化処理が終わるまで待ち合わせを行います。手順を下記に示します。

- PE1 と同じく相互排他変数 MEV を 0 で初期化します。
- PE3 自身の処理完了を示す MEV のビット 1 に 1 を立てます。
- PE1 の処理完了を示す MEV のビット 0 の値を調べます。
- MEV のビット 0 の値が 1 であれば，PE1，PE3 とも初期化処理が完了したことになるので待ち合わせ処理を抜けます。0 であれば，PE1 がまだ初期化処理中なので，2 つ目の処理に戻り待ち合わせを続けます。

(7) zeroclr4 ルーチン
ECC 機能のために各 RAM を初期化するルーチンです。r6，r7 レジスタで渡されたアドレスの範囲内をゼロクリアします。

注意 初期化対象メモリ，初期化方法はデバイスのユーザーズ・マニュアルを参照してください。

(8) init_eiint ルーチン
EI レベル例外の初期化を行うサブルーチンです。以下の処理を行います。

- INTBP レジスタに EIINTTBL セクションの先頭アドレスを設定します。
- サンプルでは，割り込み制御レジスタを使用していくつかの EI レベル例外の分岐方式を拡張仕様（テーブル参照方式）に設定します。拡張仕様の例外ハンドラを使用しない場合は，設定する必要はありません。

サンプルでは無効にしてあります。有効にするにはファイル冒頭の USE_TABLE_REFERENCE_METHOD マクロを有効にしてください。

注意 割り込み制御レジスタについてはデバイスのユーザーズ・マニュアルを参照してください。

8.2.3 アプリケーション・プロジェクト用スタートアップ・ルーチン

アプリケーション・プロジェクト用スタートアップ・ルーチンは，ユーザ・アプリケーション（main 関数以降）を実行するために必要なアプリケーション単位の初期設定を行うルーチンです。

- シングルコア用のプログラムでは，vecttbl.asm，cstart.asm の 2 つのファイルを用意します。
- マルチコア用のプログラムでは，cstart.asm をアプリケーション・プロジェクトごとに用意します。
  vecttbl.asm はブート・ローダ・プロジェクトにのみ用意します。

vecttbl.asm（例外ベクタ・テーブル）は次の要素で構成されます。

- RESET ベクタ
- 割り込みハンドラ・テーブル
- 例外ハンドラ・ルーチン

備考 アプリケーション・プロジェクトからブート・ローダ・プロジェクトへ受け渡す設定情報以外は，ラベル名，セクション名とも，アプリケーション・プロジェクト間で重複してもかまいません。重複による混乱を避けるために，プロジェクトごとに固有のセクション名を使用したい場合は，コンパイラの -Xmulti_level=1 オプションを指定したうえで，C ソース上で #pragma pmodule を使用してください。

cstart.asm は次の要素で構成されます。

- スタック領域
- RAM セクション初期化テーブル
- エントリ・ルーチン
- ユーザ・アプリケーションへの分岐処理ルーチン
- abort ルーチン
- hdwinit ルーチン
- zeroclr4 ルーチン
- init_eiint ルーチン
- アプリケーション情報テーブル

(1) スタック領域
コンパイラ生成コードが利用するスタック領域です。
サンプルでは，スタック領域は bss 属性の .stack.bss セクションに確保しています。

備考 .bss セクション以外に確保する目的は，ユーザ変数領域と配置位置を区別するためと，C 言語仕様で初期化が省略可能なことからプログラム起動時間の短縮を図るためです。

CC-RH は，スタック・ポインタ（sp）相対でメモリを参照する場合，sp が 4 バイト境界に位置していることを前提としたコードを出力し，スタック領域をアドレスの 0x0 番地方向に成長させます。そのため，.stack.bss セクションの 0xffffffff 番地側の端が 4 バイトに整列している必要があります。

(2) RAM セクション初期化テーブル
C ソース・プログラムで外部変数を使用している場合，プログラム実行前にその領域を初期化する必要があります。
初期化は外部変数を配置したセクション単位で行い，最適化リンカの生成するセクション先頭，終端ラベルを利用します。
初期値ありセクションの場合，以下のように “#__s セクション名 , #__e セクション名 , #__s 初期化先セクション名 ” の並びでテーブルの 1 行を構成し，必要なセクションの数だけこの行を配置します。
初期値ありセクションの初期化テーブルには以下の規則があります。

- .INIT_DSEC.const セクションに配置する必要があり，セクション名は固定です。
- 初期化テーブル以外を配置することはできません。
- 初期化テーブルの先頭は 4 バイトに整列している必要があります。

```
.section    ".INIT_DSEC.const", const
.align      4
.dw         #__s.data, #__e.data, #__s.data.R
.dw         #__s.sdata, #__e.sdata, #__s.sdata.R
```

初期化先セクションの生成には最適化リンカの -rom オプションを使用します。

```
-rom=.data=.data.R
-rom=.sdata=.sdata.R
```

初期値なしセクションの場合，以下のように “#__s セクション名 , #__e セクション名 ” の並びでテーブルの 1 行を構成し，必要なセクションの数だけこの行を配置します。
初期値なしセクションの初期化テーブルには以下の規則があります。

- .INIT_BSEC.const セクションに配置する必要があり，セクション名は固定です。
- 初期化テーブル以外を配置することはできません。
- 初期化テーブルの先頭は 4 バイトに整列している必要があります。

```
.section    ".INIT_BSEC.const", const
.align      4
.dw         #__s.bss, #__e.bss
.dw         #__s.sbss, #__e.sbss
```

これらのテーブルに記述してあるセクションがユーザ・アプリケーションに存在しない場合，リンク時にエラーとなります。これを回避するために，テーブルに記述したセクション名に対応する以下のような空のダミー・セクションを用意します。

```
        .section    ".data", data
.L.dummy.data:
        .section    ".sdata", sdata
.L.dummy.sdata:
        .section    ".bss", bss
.L.dummy.bss:
        .section    ".sbss", sbss
.L.dummy.sbss:
```

マルチコア用のプログラムで #pragma pmodule を使用している場合は，自動的にセクション名が修飾されるため，以下のように現れる可能性のあるセクション名すべてを記述します。

```
        .section    ".INIT_DSEC.const", const
        .align      4
        .dw         #__s.data.pm1, #__e.data.pm1, #__s.data.pm1.R
        .dw         #__s.data.cmn, #__e.data.cmn, #__s.data.cmn.R
        .dw         #__s.data, #__e.data, #__s.data.R
        .dw         #__s.sdata.pm1, #__e.sdata.pm1, #__s.sdata.pm1.R
        .dw         #__s.sdata.cmn, #__e.sdata.cmn, #__s.sdata.cmn.R
        .dw         #__s.sdata, #__e.sdata, #__s.sdata.R

        .section    ".INIT_BSEC.const", const
        .align      4
        .dw         #__s.bss.pm1, #__e.bss.pm1
        .dw         #__s.bss.cmn, #__e.bss.cmn
        .dw         #__s.bss, #__e.bss
        .dw         #__s.sbss.pm1, #__e.sbss.pm1
        .dw         #__s.sbss.cmn, #__e.sbss.cmn
        .dw         #__s.sbss, #__e.sbss

        .section    ".data.pm1", data
.L.dummy.data.pm1:
        .section    ".data.cmn", data
.L.dummy.data.cmn:
        .section    ".data", data
.L.dummy.data:
        .section    ".sdata.pm1", sdata

.L.dummy.data.pm1:
        .section    ".sdata.cmn", sdata
.L.dummy.data.cmn:
        .section    ".sdata", sdata
.L.dummy.data:

        .section    ".bss.pm1", bss
.L.dummy.bss.pm1:
        .section    ".bss.cmn", bss
.L.dummy.bss.cmn:
        .section    ".bss", bss
.L.dummy.bss:

        .section    ".sbss.pm1", sbss
.L.dummy.bss.pm1:
        .section    ".sbss.cmn", sbss
.L.dummy.bss.cmn:
        .section    ".sbss", sbss
.L.dummy.sbss:
```

備考　.INIT_DSEC.const，.INIT_BSEC.const セクションの内容に過不足がなければ，ダミー・セクションは不要です。

(3) エントリ・ルーチン
以下の手順で実行環境（PE）の初期化を行います。

(a) sp（r3）レジスタへスタック領域の最上位アドレスを設定します。設定値は必ず 4 の倍数でなければいけません。

(b) gp（r4）レジスタへ __gp_data ラベルのアドレスを設定します。__gp_data ラベルは，gp 相対セクションを使用している場合に，最適化リンカが自動生成するラベルです。

(c) ep（r30）レジスタへ __ep_data ラベルのアドレスを設定します。__ep_data ラベルは，ep 相対セクションを使用している場合に，最適化リンカが自動生成するラベルです。

備考　プログラム中で gp 相対セクション，ep 相対セクションを使用していない場合に，gp，ep の設定コードがあっても冗長なだけで問題は起きません。

(d) hdwinit ( ) を呼び出して PE に固有な初期化を行います。

ブート・ローダ・プロジェクトでも同様の初期化が可能ですが，主にアプリケーション側で定義したデータにもとづく初期化をアプリケーション・プロジェクト側で行い，PE 間で共有するデータによる初期化をブート・ローダ・プロジェクト側で行います。

(e) _INITSCT_RH ( ) を呼び出して RAM セクションの初期化を行います。

- 初期値あり，初期値なしセクションの両方を初期化する場合

```
mov     #__s.INIT_DSEC.const, r6
mov     #__e.INIT_DSEC.const, r7
mov     #__s.INIT_BSEC.const, r8
mov     #__e.INIT_BSEC.const, r9
jarl    __INITSCT_RH, lp
```

- 初期値ありセクションのみを初期化する場合

```
mov     #__s.INIT_DSEC.const, r6
mov     #__e.INIT_DSEC.const, r7
mov     r0, r8
mov     r0, r9
jarl    __INITSCT_RH, lp
```

- 初期値なしセクションのみを初期化する場合

```
mov     r0, r6
mov     r0, r7
mov     #__s.INIT_BSEC.const, r8
mov     #__e.INIT_BSEC.const, r9
jarl    __INITSCT_RH, lp
```

(f) ユーザ・アプリケーションへの分岐処理へ分岐します。

(4) ユーザ・アプリケーションへの分岐処理ルーチン
以下の手順でユーザ・アプリケーションへの分岐を行います。

(a) FEPSW レジスタを経由して PSW の設定とユーザ・アプリケーションへ分岐するための分岐先の設定を行います。
サンプルでは，それぞれ以下の設定を行っています。

対象プログラム	FPU	EI レベル例外	ユーザ・モード
シングルコア用のプログラム（R7F7010352AFP 用）	OFF	OFF	OFF
マルチコア用のプログラム（R7F701Z07 用）	ON	OFF	ON

(b) FEPC レジスタにユーザ・アプリケーション先頭アドレス（#_main）を設定します。

(c) feret 命令を実行して，PSW 設定の反映とユーザ・アプリケーションへの分岐を行います。

備考 CPU の動作モードを変更しない場合は，feret 命令ではなく分岐命令を使用しても問題はありません。

```
jarl32  #_main, lp
```

(5) abort ルーチン
ユーザ・アプリケーションで assert マクロ，または abort 自体を使用する場合に必要になります。サンプルでは自分自身への分岐を繰り返すのみとしています。

(6) hdwinit ルーチン
シングルコア用のプログラム用のサンプルでは ECC 機能向けに Local RAM を初期化します。

注意 サンプルでは R7F7010352AFP を対象としたアドレス範囲を指定しています。
デバイスのユーザーズ・マニュアルを参照して適切なアドレスを指定してください。

(7) zeroclr4 ルーチン
ECC 機能のために各 RAM を初期化するルーチンです。r6，r7 レジスタで渡されたアドレスの範囲内をゼロクリアします。

注意 初期化対象メモリ，初期化方法はデバイスのユーザーズ・マニュアルを参照してください。

(8) init_eiint ルーチン
EI レベル例外の初期化を行うサブルーチンです。以下の処理を行います。

- INTBP レジスタに EIINTTBL セクションの先頭アドレスを設定します。
- サンプルでは，割り込み制御レジスタを使用していくつかの EI レベル例外の分岐方式を拡張仕様（テーブル参照方式）に設定します。拡張仕様の例外ハンドラを使用しない場合は，設定する必要はありません。

サンプルでは無効にしてあります。有効にするにはファイル冒頭の USE_TABLE_REFERENCE_METHOD マクロを有効にしてください。

注意 割り込み制御レジスタについてはデバイスのユーザーズ・マニュアルを参照してください。

(9) アプリケーション情報テーブル
マルチコア用のプログラムでブート・ローダ・プロジェクトに受け渡したいコンテキスト情報を格納したテーブルです。
サンプルでは以下の情報をテーブルに格納しています。

オフセット	値	備考
0	アプリケーションのエントリ・ルーチン・アドレス	__start に相当するラベルの絶対アドレスを指定

テーブルの配置先セクション名は .const.cmn としています。このセクション内のラベルをブート・ローダ・プロジェクトから参照しています。

注意 スタートアップ・ルーチンをアプリケーション・プロジェクト間でコピーする際，.const.cmn 内のラベル名が重複しないように変更する必要があります。

備考 アプリケーション情報テーブルは，必要に応じて変更することができます。変更する場合は，ブート・ローダ・プロジェクト側のテーブルを読み出す処理も対応する必要があります。

8.2.4 アプリケーション・プロジェクトからブート・ローダ・プロジェクトへの情報の受け渡し

マルチコア用のプログラムは，ブート・ローダ・プロジェクトと複数のアプリケーション・プロジェクトから構成されます。
ブート・ローダ・プロジェクトからハードウエアの初期化や各アプリケーション・プロジェクトで設定した情報を利用するために，最適化リンカの -fsymbol オプションを使用します。これによって，指定したセクションに存在する public ラベルの名前と，リンク後のアドレス値の情報をオプションを使用してファイル（シンボル・アドレス・ファイル）に出力させ，ブート・ローダ・プロジェクトへ受け渡せるようになります。
アプリケーション・プロジェクトのエントリ・ルーチン・アドレスをブート・ローダ・プロジェクトに受け渡す例を以下に示します。

__start_pm1 のアドレスをブート・ローダ・プロジェクトから参照するため，__start_pm1 を public 指定します。

```
;; cstart.asm
        .section    ".text.cmn", text
        .public     __start_pm1
__start_pm1:
        jarl        _main, lp
```

アプリケーション・プロジェクトのリンク時に，-fsymbol オプションで __start_pm1 ラベルの存在する .text.cmn セクションを指定します。この場合，pm1.fsy ファイルが生成されます。

```
>rlink cstart.obj .opm1.abs -fsymbol=.text.cmn
```

ブート・ローダ・プロジェクトで __start_pm1 を参照します。

```
;; boot.asm
        .section    ".text", text
__start:
        cmp         1, r6
        bnz         .L1
        jr32        #__start_pm1
.L1:
```

ブート・ローダ・プロジェクトのコンパイル時に，各アプリケーション・プロジェクトから生成した .fsy ファイルを一緒にコンパイルすると，.fsy ファイル内のアドレス値で boot.asm 内の参照解決が行われます。

```
>ccrh boot.asm pm1.fsy pm2.fsy -oboot.abs
```

注意 -fsymbol オプションで指定したセクション内の public ラベルは，すべてブート・ローダ・プロジェクト内で定義されているとみなされます。そのため，異なるアプリケーション・プロジェクトでそれぞれの指定セクション内に同名のラベルがあると，ブート・ローダ側のリンク時に多重定義エラーになります（セクション名はアプリケーション間で同名でも衝突しません）。
したがって，アプリケーション・プロジェクトで -fsymbol オプションに指定するセクションには多くのラベルを定義しない，定義する場合は他のアプリケーション・プロジェクトと衝突しないラベル名を付けるなどの注意が必要です。

8.3 コーディング例

例外ベクタテーブルの例を，次に示します。

表 8.1 例外ベクタテーブルの例

```
        .section        "RESET", text
        .align  512
        jr32    __start         ; RESET

        .align  16
        jr32    _Dummy          ; SYSERR

        .align  16
        jr32    _Dummy          ; HVTRAP

        .align  16
        jr32    _Dummy          ; FETRAP

        .align  16
        jr32    _Dummy_EI       ; TRAP0

        .align  16
        jr32    _Dummy_EI       ; TRAP1

        .align  16
        jr32    _Dummy          ; RIE

        .align  16
        jr32    _Dummy_EI       ; FPP/FPI

        .align  16
        jr32    _Dummy          ; UCPOP

        .align  16
        jr32    _Dummy          ; MIP/MDP/ITLBE/DTLBE

        .align  16
        jr32    _Dummy          ; PIE

        .align  16
        jr32    _Dummy          ; Debug

        .align  16
        jr32    _Dummy          ; MAE
```

```
        .align  16
        jr32    _Dummy          ; (R.F.U)

        .align  16
        jr32    _Dummy          ; FENMI


        .align  16
        jr32    _Dummy          ; FEINT

        .align  16
        jr32    _Dummy_EI       ; INTn(priority0)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority1)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority2)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority3)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority4)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority5)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority6)

        .align  16
        jr32    _Dummy_EI       ; INTn(priority7)

        .section        "EIINTTBL", const
        .align  512
        .dw     #_Dummy_EI      ; INT0
        .dw     #_Dummy_EI      ; INT1
        .dw     #_Dummy_EI      ; INT2
        .rept   288 - 3
        .dw     #_Dummy_EI      ; INTn
        .endm

        .section        ".text", text
        .align  2
_Dummy:
        br      _Dummy

_Dummy_EI:
        br      _Dummy_EI
```

シングルコア用プロジェクトのスタートアップ・ルーチンの例を，次に示します。

表 8.2 シングルコア用プロジェクトのスタートアップ・ルーチンの例

```
; NOTE  : THIS IS A TYPICAL EXAMPLE. (R7F7010352AFP)

        ; example of using eiint as table refrence method
        ;USE_TABLE_REFERENCE_METHOD .set 1

;-------------------------------------------------------------------------------
;       system stack
;-------------------------------------------------------------------------------
STACKSIZE           .set    0x200
        .section    "stack.bss", bss
        .align      4
        .ds         (STACKSIZE)
        .align      4
.stacktop:

;-------------------------------------------------------------------------------
;       section initialize table
;-------------------------------------------------------------------------------
        .section    ".INIT_DSEC.const", const
        .align      4
        .dw         #__s.data, #__e.data, #__s.data.R

        .section    ".INIT_BSEC.const", const
        .align      4
        .dw         #__s.bss, #__e.bss

;-------------------------------------------------------------------------------
;       startup
;-------------------------------------------------------------------------------
        .section    ".text", text
        .public     __start
        .align      2
__start:
        mov         #_stacktop, sp          ; set sp register
        mov         #__gp_data, gp          ; set gp register
        mov         #__ep_data, ep          ; set ep register
```

```
        jarl        _hdwinit, lp            ; initialize hardware

        mov         #__s.INIT_DSEC.const, r6
        mov         #__e.INIT_DSEC.const, r7
        mov         #__s.INIT_BSEC.const, r8
        mov         #__e.INIT_BSEC.const, r9
        jarl32      __INITSCT_RH, lp        ; initialize RAM area

$ifdef USE_TABLE_REFERENCE_METHOD
        jarl        _init_eiint, lp         ; initialize exception
$endif

        ; set various flags to PSW via FEPSW
        stsr        5, r10, 0               ; r10 <- PSW

        ;xori       0x0020, r10, r10        ; enable interrupt

        ;movhi      0x4000, r0, r11
            ;or         r11, r10            ; supervisor mode -> user mode

        ldsr        r10, 3, 0               ; FEPSW <- r10

        mov         #_exit, lp              ; lp <- #_exit
        mov         #_main, r10
        ldsr        r10, 2, 0               ; FEPC <- #_main

        ; apply PSW and call main
        feret
_exit:
        br          _exit                   ; end of program

;-------------------------------------------------------------------------------
;      abort
;-------------------------------------------------------------------------------
        .public     _abort
        .align      2
_abort:
        br          _abort

;-------------------------------------------------------------------------------
;      target dependence informations (specify values suitable to your system)
;-------------------------------------------------------------------------------
        ; RAM address
        PRIMARY_LOCAL_RAM_ADDR      .set    0xfede0000
```

```
        PRIMARY_LOCAL_RAM_END       .set    0xfedfffff

        SECONDARY_LOCAL_RAM_ADDR    .set    0xfedd8000
        SECONDARY_LOCAL_RAM_END     .set    0xfeddffff

        RETENTION_RAM_ADDR          .set    0xfee00000
        RETENTION_RAM_END           .set    0xfee07fff

;-------------------------------------------------------------------------------
;      hdwinit
;-------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit:
        mov         lp, r14                 ; save return address

        mov         PRIMARY_LOCAL_RAM_ADDR, r6
        mov         PRIMARY_LOCAL_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Primary local RAM

        mov         SECONDARY_LOCAL_RAM_ADDR, r6
        mov         SECONDARY_LOCAL_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Secondary local RAM

        mov         RETENTION_RAM_ADDR, r6
        mov         RETENTION_RAM_END, r7
        jarl        _zeroclr4, lp           ; clear Retention RAM

        mov         r14, lp
        jmp         [lp]

;-------------------------------------------------------------------------------
;      zeroclr4
;-------------------------------------------------------------------------------
        .align      2
_zeroclr4:
        br          .L.zeroclr4.2
.L.zeroclr4.1:
        st.w        r0, [r6]
        add         4, r6
.L.zeroclr4.2:
        cmp         r6, r7
        bh          .L.zeroclr4.1
        jmp         [lp]
```

```
$ifdef USE_TABLE_REFERENCE_METHOD
;-------------------------------------------------------------------------------
;       init_eiint
;-------------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE                      .set    0xffff9000
        .align      2
_init_eiint:
        mov         #__sEIINTTBL, r10
        ldsr        r10, 4, 1               ; set INTBP
        ; some inetrrupt channels use the table reference method.
        mov         ICBASE, r10             ; get interrupt control register address
        set1        6, 0[r10]               ; set INT0 as table reference
        set1        6, 2[r10]               ; set INT1 as table reference
        set1        6, 4[r10]               ; set INT2 as table reference
        jmp         [lp]
$endif

;-------------------------------------------------------------------------------
;       dummy section
;-------------------------------------------------------------------------------
        .section    ".data", data
.L.dummy.data:
        .section    ".bss", bss
.L.dummy.bss:
        .section    ".const", const
.L.dummy.const:
        .section    ".text", text
.L.dummy.text:
;-------------------- end of start up module -------------------;
```

マルチコア用ブート・ローダ・プロジェクトのスタートアップ・ルーチンの例を，次に示します。

表 8.3　マルチコア用ブート・ローダ・プロジェクトのスタートアップ・ルーチンの例

```
; NOTE  : THIS IS A TYPICAL EXAMPLE. (R7F701Z07)

        ; example of using eiint as table reference method
        ;USE_TABLE_REFERENCE_METHOD .set 1

        ; offset of processing module setting table element
        .OFFSET_ENTRY       .set    0

;-------------------------------------------------------------------------------
;       startup
```

```
;-------------------------------------------------------------------------------
        .section    ".text", text
        .public     __start
        .align      2
__start:
        ; jump to entry point of each PE
        stsr        0, r10, 2               ; get HTCFG0
        shr         16, r10                 ; get PEID

        cmp         1, r10
        bz          __start_PE1
        cmp         3, r10
        bz          __start_PE3
__exit:
        br          __exit

__start_PE1:
        jarl        _hdwinit_PE1, lp        ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        jarl        _init_eiint, lp         ; initialize exception
$endif

        mov         #_pm1_setting_table, r13
        ld.w        .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        ldsr        r10, 2, 0               ; set FEPC

        ; apply PSW and jump to the application entry point
        feret

__start_PE3:
        jarl        _hdwinit_PE3, lp        ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
        jarl        _init_eiint, lp         ; initialize exception
$endif
        ;mov        #_pm2_setting_table, r13
        ;ld.w       .OFFSET_ENTRY[r13], r10 ; r10 <- #__start
        ;ldsr       r10, 2, 0               ; set FEPC

        ; apply PSW and jump to the application entry point
        ;feret
        br          __exit

;-------------------------------------------------------------------------------
;      target dependence informations (specify values suitable to your system)
;-------------------------------------------------------------------------------
```

```
$if 1
        ; RAM address
        GLOBAL_RAM_ADDR             .set    0xfeef0000
        GLOBAL_RAM_END              .set    0xfef1ffff

        LOCAL_RAM_PE1_ADDR          .set    0xfedf0000
        LOCAL_RAM_PE1_END           .set    0xfedfffff

        LOCAL_RAM_PE3_ADDR          .set    0xfedf8000
        LOCAL_RAM_PE3_END           .set    0xfedfffff

        ; mutual exclusion variable
        MEV_ADDR                    .set    0xfeef0000

;-------------------------------------------------------------------------------
;      hdwinit_PE1
;-------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit_PE1:
        mov         lp, r14                 ; save return address

        ; clear Global RAM
        mov         GLOBAL_RAM_ADDR, r6
        mov         GLOBAL_RAM_END, r7
        jarl        _zeroclr4, lp

        ; clear Local RAM PE1
        mov         LOCAL_RAM_PE1_ADDR, r6
        mov         LOCAL_RAM_PE1_END, r7
        jarl        _zeroclr4, lp

        ; wait for PE3
        mov         MEV_ADDR, r10
        st.w        r0, [r10]
.L.hdwinit_PE1.1:
        snooze
        set1        0, [r10]
        tst1        1, [r10]
        bz          .L.hdwinit_PE1.1

        mov         r14, lp
        jmp         [lp]

;-------------------------------------------------------------------------------
;      hdwinit_PE3
```

```
;------------------------------------------------------------------------------
        .section    ".text", text
        .align      2
_hdwinit_PE3:
        mov         lp, r14                 ; save return address

        ; clear Local RAM PE3
        mov         LOCAL_RAM_PE3_ADDR, r6
        mov         LOCAL_RAM_PE3_END, r7
        jarl        _zeroclr4, lp

        ; wait for PE1
        mov         MEV_ADDR, r10
        st.w        r0, [r10]
.L.hdwinit_PE3.1:
        snooze
        set1        1, [r10]
        tst1        0, [r10]
        bz          .L.hdwinit_PE3.1

        mov         r14, lp
        jmp         [lp]

;------------------------------------------------------------------------------
;      zeroclr4
;------------------------------------------------------------------------------
        .align      2
_zeroclr4:
        br          .L.zeroclr4.2
.L.zeroclr4.1:
        st.w        r0, [r6]
        add         4, r6
.L.zeroclr4.2:
        cmp         r6, r7
        bh          .L.zeroclr4.1
        jmp         [lp]
$endif

$ifdef USE_TABLE_REFERENCE_METHOD
;------------------------------------------------------------------------------
;      init_eiint
;------------------------------------------------------------------------------
        ; interrupt control register address
        ICBASE .set 0xfffeea00
```

```
        .align 2
_init_eiint:
        mov         #__sEIINTTBL, r10
        ldsr        r10, 4, 1               ; set INTBP

        ; some inetrrupt channels use the table reference method.
        mov         ICBASE, r10             ; get interrupt control register address
        set1        6, 0[r10]               ; set INT0 as table reference
        set1        6, 2[r10]               ; set INT1 as table reference
        set1        6, 4[r10]               ; set INT2 as table reference

        jmp         [lp]
$endif
;-------------------- end of start up module --------------------;
```

マルチコア用アプリケーション・プロジェクトのスタートアップ・ルーチンの例を，次に示します。

表 8.4 マルチコア用アプリケーション・プロジェクトのスタートアップ・ルーチンの例

```
; NOTE  : THIS IS A TYPICAL EXAMPLE. (R7F701Z07)

;-------------------------------------------------------------------------------
;      processing module setting table
;-------------------------------------------------------------------------------
        .section    ".const.cmn", const
        .public     _pm1_setting_table
        .align      4
_pm1_setting_table:
        .dw         #__start                ; ENTRY ADDRESS

;-------------------------------------------------------------------------------
;      system stack
;-------------------------------------------------------------------------------
        STACKSIZE                   .set    0x200
        .section    ".stack.bss", bss
        .align      4
        .ds         (STACKSIZE)
        .align      4
_stacktop:

;-------------------------------------------------------------------------------
;      section initialize table
;-------------------------------------------------------------------------------
        .section    ".INIT_DSEC.const", const
        .align      4
```

```
        .dw         #__s.data, #__e.data, #__s.data.R

        .section    ".INIT_BSEC.const", const
        .align      4
        .dw         #__s.bss, #__e.bss

;-------------------------------------------------------------------------------
;      startup
;-------------------------------------------------------------------------------
        .section    ".text", text
        .public     __start
        .align      2
__start:
        mov         #_stacktop, sp          ; set sp register
        mov         #__gp_data, gp          ; set gp register
        mov         #__ep_data, ep          ; set ep register

        jarl        _hdwinit, lp            ; initialize hardware

        mov         #__s.INIT_DSEC.const, r6
        mov         #__e.INIT_DSEC.const, r7
        mov         #__s.INIT_BSEC.const, r8
        mov         #__e.INIT_BSEC.const, r9
        jarl32      __INITSCT_RH, lp        ; initialize RAM area

        ; set various flags to PSW via FEPSW

        stsr        5, r10, 0               ; r10 <- PSW

        movhi       0x0001, r0, r11
        or          r11, r10                ; enable FPU

        ;xori       0x0020, r10, r10        ; enable interrupt

        movhi       0x4000, r0, r11
        or          r11, r10                ; supervisor mode -> user mode

        ldsr        r10, 3, 0               ; FEPSW <- r10

        mov         #_exit, lp              ; lp <- #_exit
        mov         #_main, r10
        ldsr        r10, 2, 0               ; FEPC <- #_main

        ; apply PSW and PC to start user mode
        feret
```

```
_exit:
        br          _exit                   ; end of program

;-------------------------------------------------------------------------------
;       abort
;-------------------------------------------------------------------------------
        .public     _abort
        .align      2
_abort:
        br          _abort

;-------------------------------------------------------------------------------
;       dummy section
;-------------------------------------------------------------------------------
        .section    ".data", data
.L.dummy.data:
        .section    ".bss", bss
.L.dummy.bss:
        .section    ".const", const
.L.dummy.const:
        .section    ".text", text
.L.dummy.text:
;-------------------- end of start up module -------------------;
```

8.4 シンボル

CC-RH では，アプリケーションを実行する際に次のポインタを必要とします。

- グローバル・ポインタ（gp）
- エレメント・ポインタ（ep）

ここでは，各ポインタの役割と，ポインタ値の決め方について説明します。

8.4.1 グローバル・ポインタ（gp）

アプリケーション内でグローバル宣言したデータはメモリに配置されます。このメモリに配置されているデータを参照（ロード，ストア）する際，配置位置に依存することのない参照（Position Independent Data：PID）を実現するために用意されるポインタが「グローバル・ポインタ（gp）」です。

(1) グローバル・ポインタ（gp）の設定
グローバル・ポインタ（gp）に設定するアドレス値は，グローバル・ポインタ・シンボル（gp シンボル）の値をアドレス値として設定します。

- gp シンボルは，固定シンボル名 “__gp_data” で扱います。
- gp シンボルは，スタートアップ・ルーチンで参照シンボル（.extern）として宣言してください。
- gp シンボルは，rlink 内で外部定義シンボル（.public）を作成しアドレス値を決定します。
- gp シンボルの定義／参照に対する動作は以下のとおりです。

(a) gp シンボルが “.extern” 宣言のみ存在している場合
rlink が定義情報を作成し，自動でアドレス値を設定します。

(b) gp シンボルが定義されている場合
定義されたアドレス値を使用します。

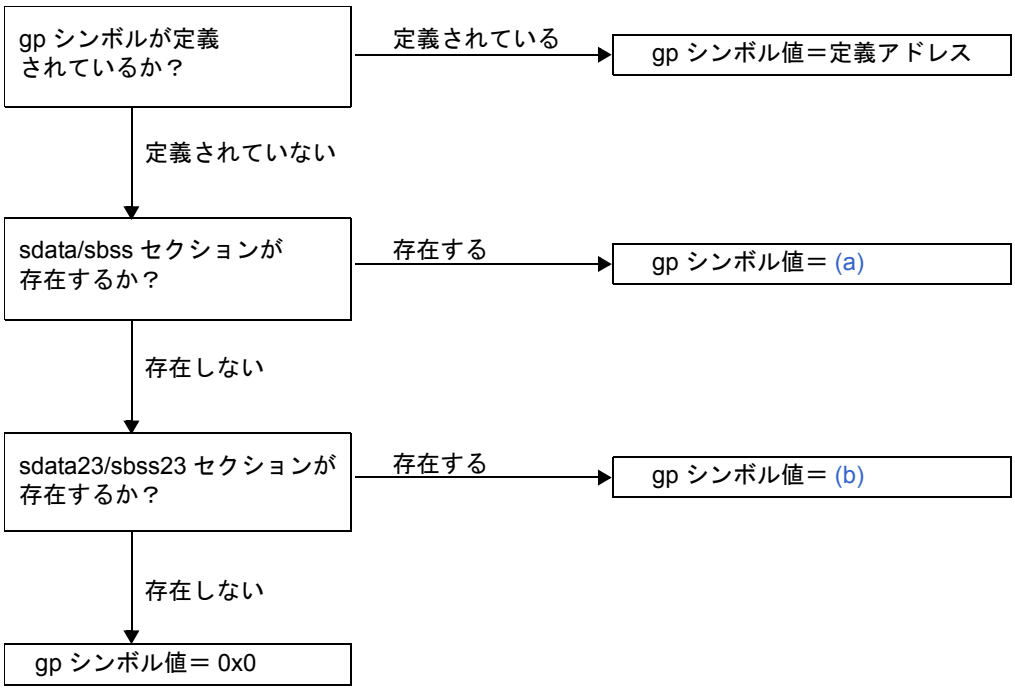
(c) gp シンボルが存在しない（使用されていない）場合
gp 相対を参照するコードに対して，次のエラー・メッセージが出力されます。

```
Undefined external symbol "GP-symbol (__gp_data)" referenced in "FILE"
```

(2) gp シンボル値の決定規則

gp シンボル値は，再配置属性が sdata，sbss，sdata23，および sbss23 のセクションから以下の優先順位で決定します。

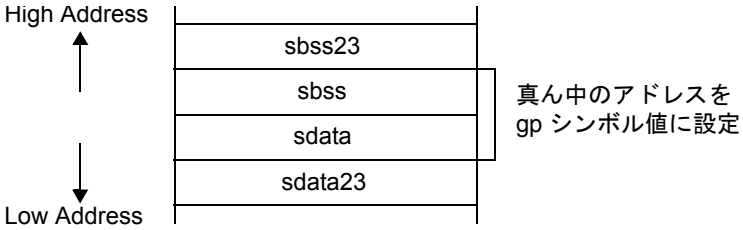
図 8.1 グローバル・ポインタ値の決定規則



以下に，gp シンボル値の決定例を示します。

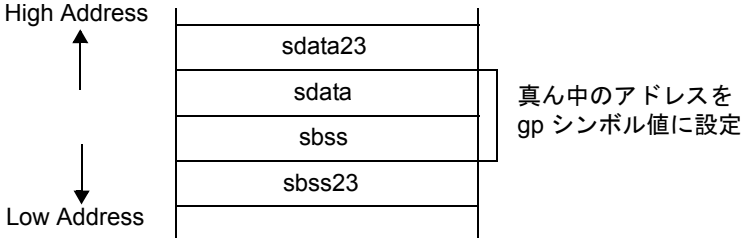
(a) 再配置属性が sdata，または sbss のセクションが存在する場合
sdata，または sbss セクションに対して，“1 番小さいアドレスに配置されたセクションの開始アドレス ” と “1 番大きいアドレスに配置されたセクションの終了アドレス ” の真ん中（2 の倍数値，奇数の場合は最初の 2 の倍数値）を gp シンボルのアドレス値として設定します。

<1> アドレスの小さい方から，sdata，sbss セクションの順で配置されている場合



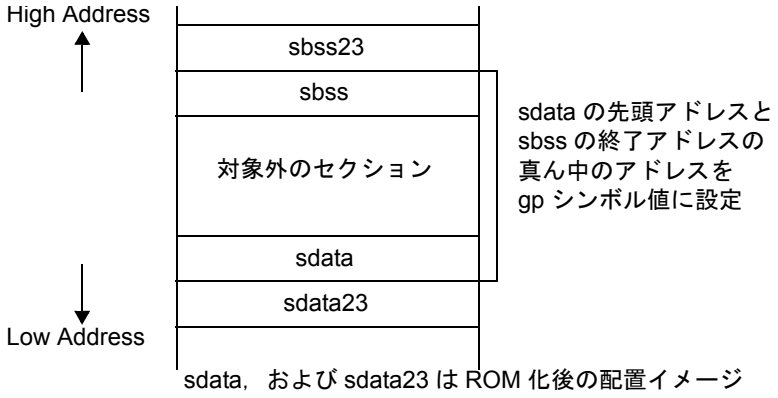
sdata，および sdata23 は ROM 化後の配置イメージ

<2> アドレスの小さい方から，sbss，sdata セクションの順で配置されている場合



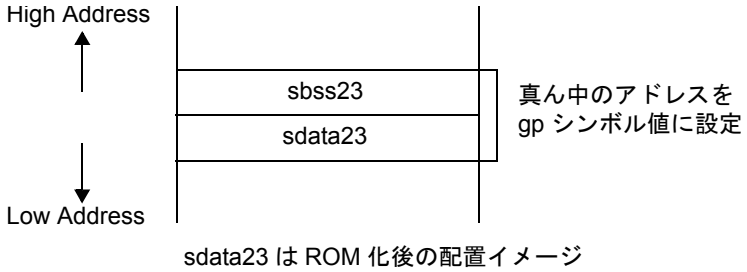
sdata，および sdata23 は ROM 化後の配置イメージ

<3> アドレスの小さい方から，sdata，sbss セクションの順で配置し，sdata，sbss セクションの間に対象外のセクションが存在している場合

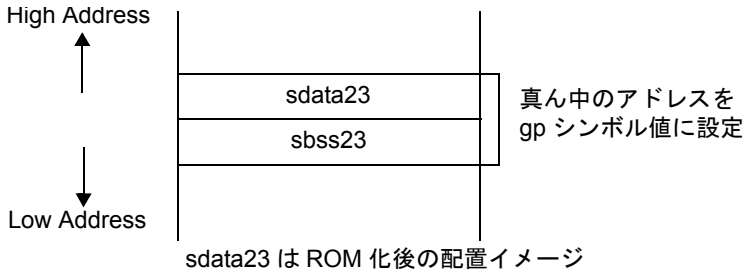


(b) 再配置属性が sdata23，または sbss23 のセクションが存在する場合
sdata23，または sbss23 セクションに対して，“1 番小さいアドレスに配置されたセクションの開始アドレス ” と “1 番大きいアドレスに配置されたセクションの終了アドレス ” の真ん中（2 の倍数値，奇数の場合は最初の 2 の倍数値）を gp シンボルのアドレス値として設定します。

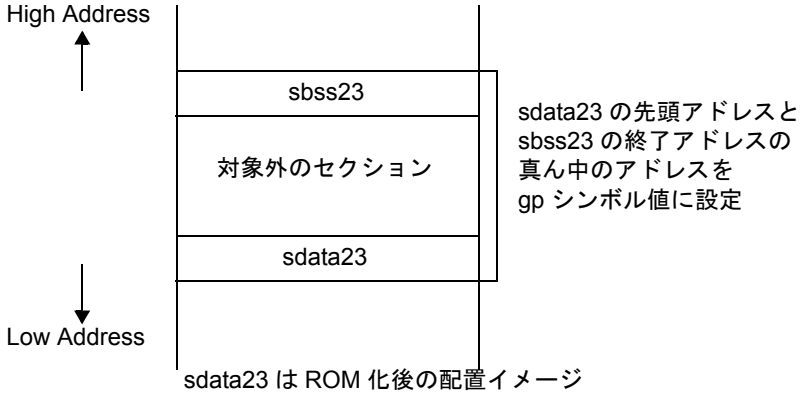
<1> アドレスの小さい方から，sdata23，sbss23 セクションの順で配置されている場合



<2> アドレスの小さい方から，sbss23，sdata23 セクションの順で配置されている場合



<3> アドレスの小さい方から，sdata23，sbss23 セクションの順で配置し，sdata23，sbss23 セクションの間に対象外のセクションが存在している場合



(c) 再配置属性が sdata，sbss，sdata23，および sbss23 のセクションが存在しない場合
sdata，sbss，sdata23，および sbss23 セクションが存在しない場合は，gp シンボルのアドレス値を 0（ゼロ）として設定します。

8.4.2 エレメント・ポインタ（ep）

アプリケーション内でグローバル宣言したデータ（変数）を，RH850 に内蔵されている RAM 領域へ配置し，より高速な参照（ロード，ストア）を実現するために用意されているポインタが「エレメント・ポインタ（ep）」です。
グローバル宣言し，かつ，内蔵 RAM に配置するデータ（変数）は ep 相対で参照されます。

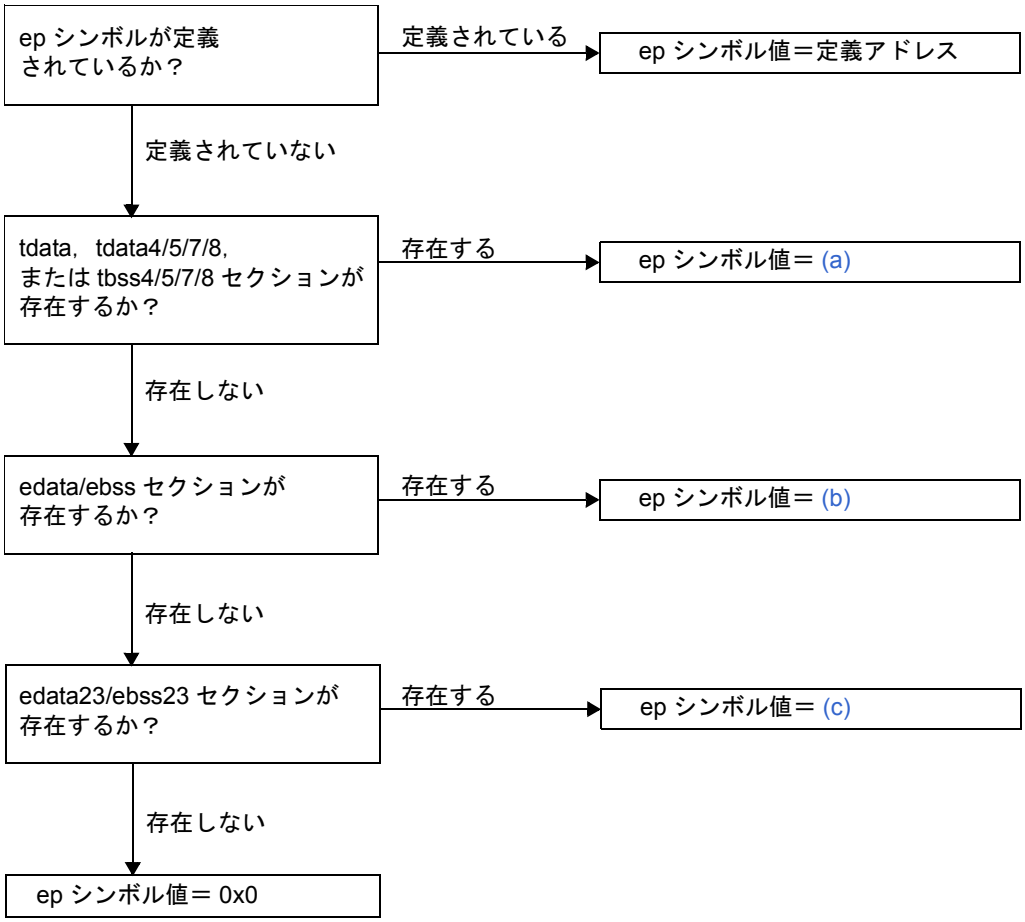
(1) エレメント・ポインタ（ep）の設定
エレメント・ポインタ（ep）に設定するアドレス値は，エレメント・ポインタ・シンボル（ep シンボル）の値をアドレス値として設定します。

- ep シンボルは，固定シンボル名 “__ep_data” で扱います。
- ep シンボルは，スタートアップ・ルーチンで参照シンボル（.extern）として宣言してください。
- ep シンボルは，rlink 内で外部定義シンボル（. public）を作成しアドレス値を決定します。
- ep シンボルの定義／参照に対する動作は以下のとおりです。
  - ep シンボルが “.extern” 宣言のみ存在している場合
    rlink が定義情報を作成し，自動でアドレス値を設定します。
  - ep シンボルが定義されている場合
    定義されたアドレス値を使用します。
  - ep シンボルが存在しない（使用されていない）場合
    ep 相対を参照するコードに対して，次のエラー・メッセージが出力されます。

```
Undefined external symbol "EP-symbol (__ep_data)" referenced in "FILE"
```

(2) ep シンボル値の決定規則
ep シンボル値は，再配置属性が tdata，tdata4/5/7/8，tbss4/5/7/8，edata，ebss，edata23，および ebss23 のセクションから以下の優先順位で決定します。

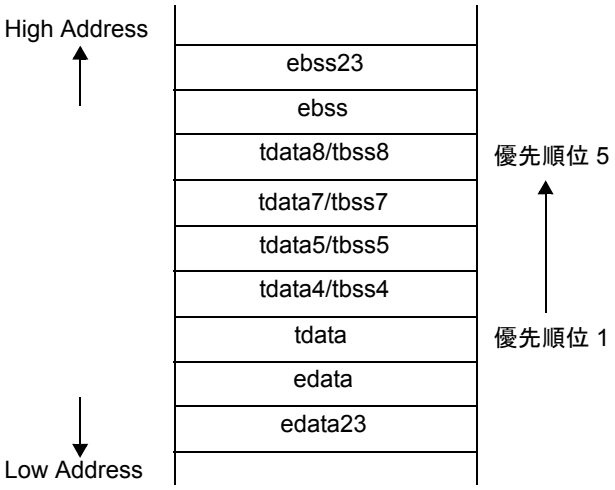
図 8.2 エレメント・ポインタ値の決定規則



以下に，ep シンボル値の決定例を示します。

(a) 再配置属性が tdata，tdata4/5/7/8，および tbss4/5/7/8 のセクションが存在する場合

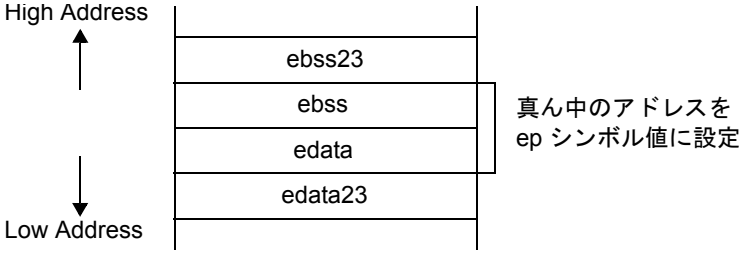
tdata → tdata4，または tbss4 →・・・→ tdata8，または tbss8 の優先順位で，存在するセクションの開始アドレスを ep シンボルに設定します。



tdata，tdata4/5/7/8，edata，および edata23 は ROM 化後の配置イメージ

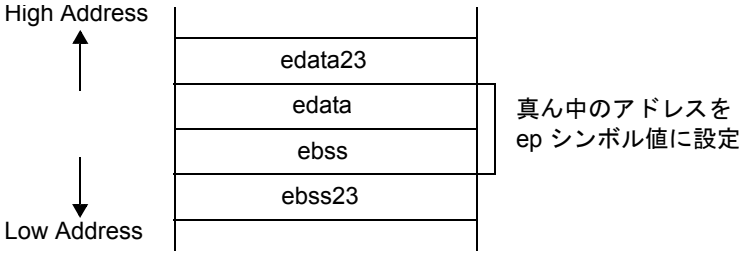
(b) 再配置属性が edata，または ebss のセクションが存在する場合
edata，または ebss セクションに対して “1 番小さいアドレスに配置されたセクションの開始アドレス ” と “1 番大きいアドレスに配置されたセクションの終了アドレス ” の真ん中（2 の倍数値，奇数の場合は最初の 2 の倍数値）を ep シンボルのアドレス値として設定します。

<1> アドレスの小さい方から，edata，ebss セクションの順で配置されている場合



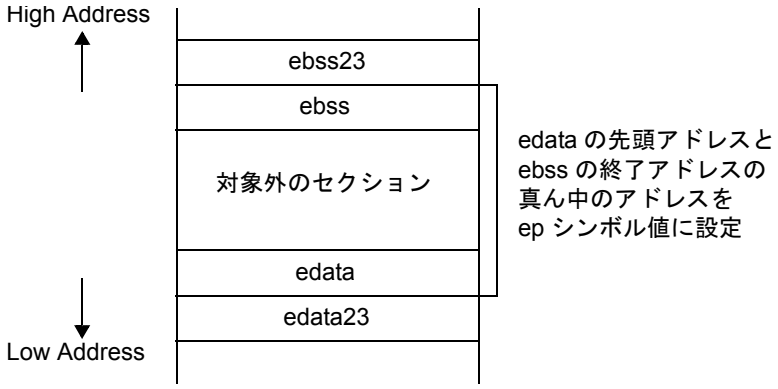
edata，および edata23 は ROM 化後の配置イメージ

<2> アドレスの小さい方から，ebss，edata セクションの順で配置されている場合



edata，および edata23 は ROM 化後の配置イメージ

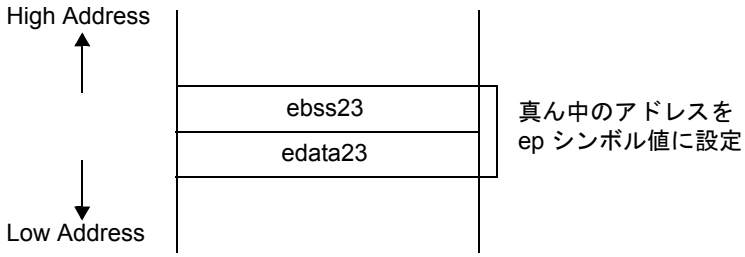
<3> アドレスの小さい方から，edata，ebss セクションの順で配置し，edata，ebss セクションの間に対象外のセクションが存在している場合



edata，および edata23 は ROM 化後の配置イメージ

(c) 再配置属性が edata23，または ebss23 のセクションが存在する場合
edata23，または ebss23 セクションに対して “1 番小さいアドレスに配置されたセクションの開始アドレス ” と “1 番大きいアドレスに配置されたセクションの終了アドレス ” の真ん中（2 の倍数値，奇数の場合は最初の 2 の倍数値）を ep シンボルのアドレス値として設定します。
以下に，ep シンボル値の決定例を示します。

<1> アドレスの小さい方から，edata23，ebss23 セクションの順で配置されている場合

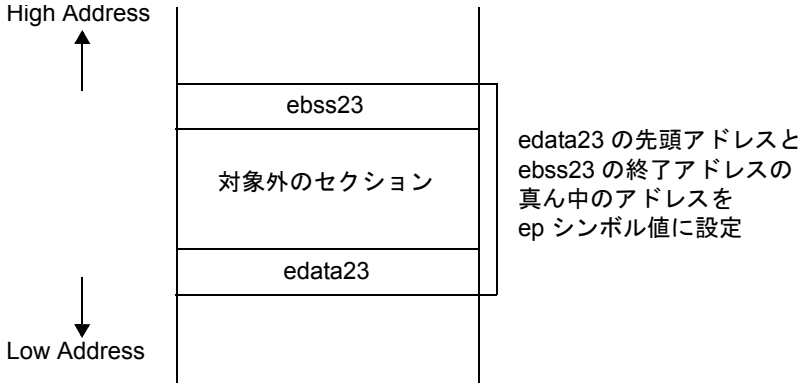


edata23 は ROM 化後の配置イメージ

<2> アドレスの小さい方から，ebss23，edata23 セクションの順で配置されている場合

edata23 は ROM 化後の配置イメージ

<3> アドレスの小さい方から，edata23，ebss23 セクションの順で配置し，edata23，ebss23 セクションの間に対象外のセクションが存在している場合



edata23 は ROM 化後の配置イメージ

(d) 再配置属性が tdata，tdata4/5/7/8，tbss4/5/7/8，edata，ebss，edata23，および ebss23 セクションが存在しない場合
tdata，tdata4/5/7/8，tbss4/5/7/8，edata，ebss，edata23，および ebss23 セクションが存在しない場合は，ep シンボルのアドレス値を 0（ゼロ）として設定します。

8.5 ROM 化

この節では，ROM 化の手順，操作方法などを説明します。

8.5.1 概　　要

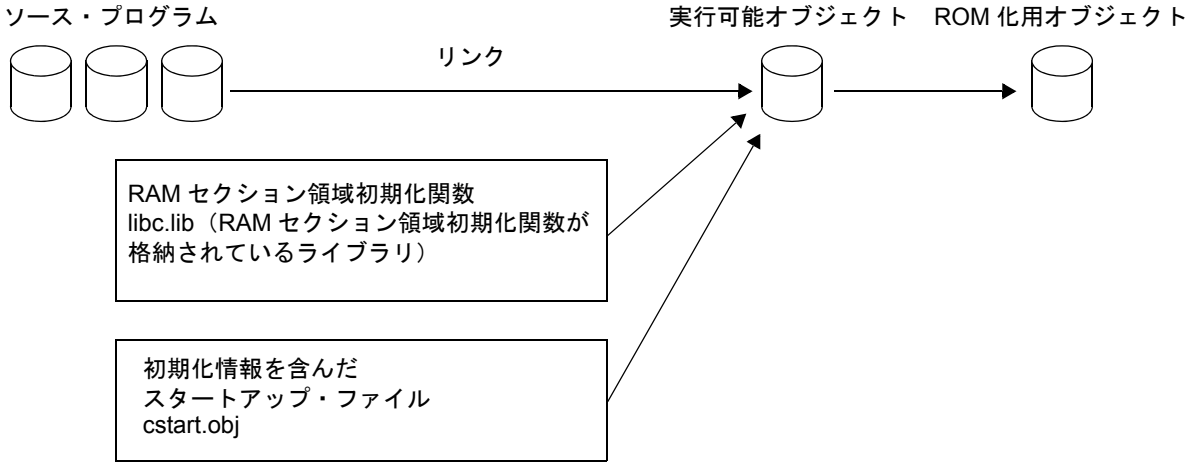
プログラム中で，グローバル，または static な変数を宣言すると，初期値を持つ変数ならば data 属性のセクションへ，初期値を持たない変数ならば bss 属性のセクションへというように，RAM 上のセクションに配置されます。特に初期値を持つ変数ならば，その初期値自体が RAM 上に配置されます。その他，アプリケーションの高速化のために，プログラム・コードを内蔵 RAM 領域へ配置する場合もあります。

組み込みシステムの場合，デバッグ時にインサーキット・エミュレータなどを使用する場合，実行可能なモジュールを配置イメージのままダウンロードして実行できます。しかし，実際にプログラムをターゲット・システムの ROM 領域に書き込んで実行する場合，data 属性のセクションにある初期値情報や，RAM 領域に配置するプログラム・コードは，実行前に RAM 上に展開されていなければなりません。つまり，RAM に展開するデータを ROM 上に持たせておき，それをアプリケーション実行前に ROM から RAM へコピーする作業が必要になります。

ROM 化は，data 属性セクションの変数の初期値情報や，RAM 上に配置するプログラムを，1 つのセクションにパッキングすることです。このセクションを ROM 上に配置し，CC-RH で用意されているコピー関数を呼び出すことによって，初期値情報やプログラムを容易に RAM 上へ展開することができます。

ROM 化用オブジェクトを作成する流れの概要は，次図のようになります。

図 8.3　ROM 化用オブジェクトの作成



「図 8.3 ROM 化用オブジェクトの作成」のように ROM 化用オブジェクトを作成すると，RAM セクション領域初期化関数 _INITSCT_RH を実行することによって，RAM に配置するデータを，パッキングされた ROM からコピーします。

ROM 領域から，RAM 領域へコピーする関数は次のとおりです。

- _INITSCT_RH

この関数はライブラリ “libc.lib” に格納されています。

また，リロケーション解決したオブジェクト・ファイルにシンボル情報，デバッグ情報が含まれる場合，それらを削除することなく ROM 化用のオブジェクト・ファイルを生成します。そのため，ROM 化後のオブジェクト・ファイルでもデバッガによるソース・デバッグができます。

8.5.2 ROM 化用ロード・モジュールの作成

この項では，ROM 化用ロード・モジュールの作成手順について説明します。

ROM 化には最適化リンカの -rom オプションを使用します。次のように記述します。

```
-rom= 初期値データセクション名 = 初期化先セクション名 [, 初期値データセクション名 = 初期化先セクション名
]...
```

また，初期値データセクション，初期化先セクションのアドレスをそれぞれ -start オプションで指定する必要があります。

```
-start=[(] セクション名 [{:|,} セクション名 ]...[)][{:|,} セクション名 ]...[/ 配置先アドレス
][,[(] セクション名 [{:|,} セクション名 ]...[)][{:|,} セクション名 ]...[/ 配置先アドレス ]]...
```

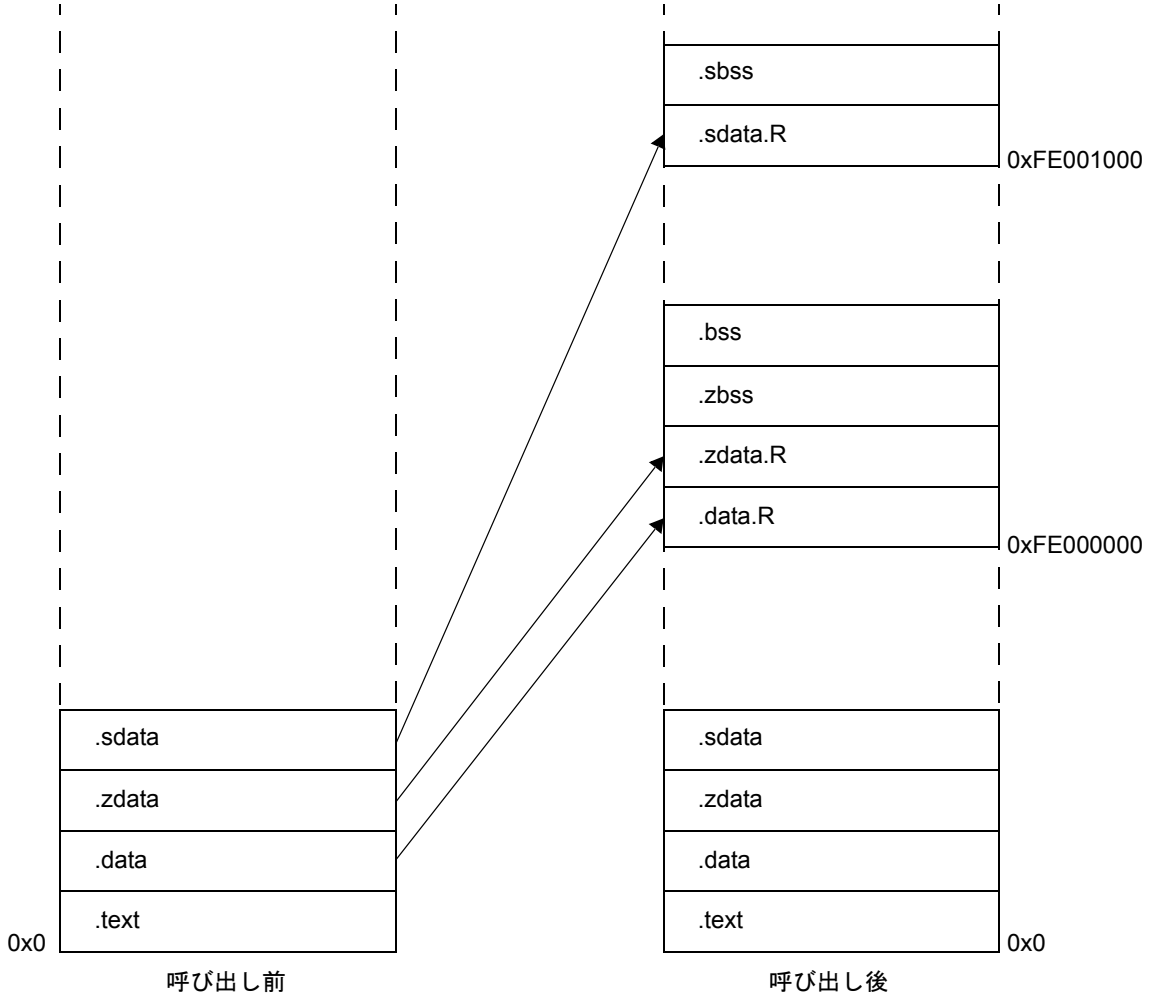
たとえば，プログラム中に .text，.data，.zdata，.zbss，.bss，.sdata，.sbss の 7 つのセクションがあり，それぞれを実行時に .text セクションは 0x0 番地，r0 相対セクションは 0xFE000000 番地，r4 相対セクションは 0xFE001000 番地に配置したい場合，次のように記述します。

```
-start=.text,.data,.zdata,.sdata/0
-start=.data.R,.zdata.R,.zbss,.bss/fe000000
-start=.sdata.R,.sbss/fe001000
-rom=.data=.data.R,.zdata=.zdata.R,.sdata=.sdata.R
```

このように記述すると，.data，.zdata は 0x0 番地の .text セクションの次に配置されます。
実行時には，0xFE000000 番地の .data.R，.zdata.R，.zbss，.bss，0xFE001000 番地の .sdata.R，.bss を初期化して使用します。

イメージは次のようになります。

図 8.4 コピー関数呼び出し前後のイメージ



9. 関数呼び出し仕様

この章では，CC-RH におけるプログラム呼び出し時の引数などの扱い方について説明します。

9.1 関数呼び出しインタフェース

この節では，CC-RH におけるプログラム呼び出し時の引数などの扱い方について説明します。

9.1.1 関数呼び出し前後で保証される汎用レジスタ

関数呼び出しの前後で，レジスタの内容が同一であることを保証する汎用レジスタと，保証しない汎用レジスタがあります。汎用レジスタの保証規則を以下に示します。

(1) 関数呼び出し前後で内容が保証されるレジスタ（Callee-Save レジスタ）
これらの汎用レジスタは，呼び出し先の関数において退避／復帰が必要なものです。そのため，呼び出し元では，関数呼び出しの前後でレジスタの中身が同一であることが保証されます。
r20，r21，r22，r23，r24，r25，r26，r27，r28，r29，r30 注，r31

注 1. r30（EP）は，プログラム全体で固定して使用する場合があります。固定して使用する場合は，プログラム全体で汎用レジスタの内容を変更しないため，呼び出し先での退避／復帰は不要となります。

(2) 関数呼び出し前後で内容が保証されないレジスタ（Caller-Save レジスタ）
上記の関数呼び出し前後で内容が保証されるレジスタ以外の汎用レジスタは，呼び出し先の関数において内容が書き換わる可能性があります。そのため，呼び出し元では，関数呼び出しの前後でレジスタの中身が同一であることが保証されません。

備考 1. r1 はアセンブラが使用する場合があるレジスタなので，書き換えはユーザ責任となります。

備考 2. r2 は OS 予約となる場合があります。予約レジスタは，コンパイラは汎用レジスタとしては使用しないので，ここに書いた規則には当てはまりません。書き換えはユーザ責任となります。

備考 3. r3 はスタック・ポインタであり，汎用レジスタの用途としては使わないのでここに書いた規則には当てはまりません。書き換えはユーザ責任となります。

備考 4. r4（GP）と r5（TP）は，プログラム全体で固定して使用する場合があります。固定して使用する場合は，プログラム全体で汎用レジスタの内容を変更しないため，ここに書いた規則には当てはまりません。書き換えはユーザ責任となります。

備考 5. r2，r30 はオプションによって使用方法を指定することがきます。

9.1.2 引数と戻り値の設定方法，および参照方法

(1) 引数の受け渡し方法
引数には，レジスタで渡すものとスタックで渡すものがあります。各引数をどちらで渡すのかは，以下の手順を実施することにより決定されます。

(a) 各引数をスタック上に割り付けたメモリ・イメージを作成する

<1> 2 バイト以下のスカラ型は，4 バイトに整数拡張して格納されます。

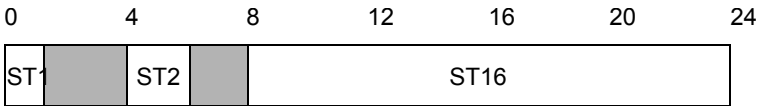
<2> 各引数は，基本的にはすべて 4 バイト境界へと配置されます。

<3> 戻り値が構造体や共用体の場合，メモリ・イメージの先頭に，戻り値データを書き込むアドレスを設定されます。

<4> 関数原型が不明な場合には，各スカラ型の引数は以下のように格納されます。

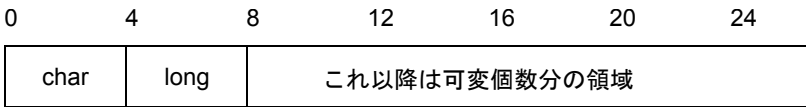
- 1 バイトのスカラ型整数 → 4 バイトに整数拡張して格納
- 2 バイトのスカラ型整数 → 4 バイトに整数拡張して格納
- 4 バイトのスカラ型整数 → そのまま格納
- 8 バイトのスカラ型整数 → そのまま格納
- 4 バイトのスカラ型浮動小数点数 → 8 バイトの浮動小数点数に拡張して格納
- 8 バイトのスカラ型浮動小数点数→ そのまま格納

例 1. 関数原型 : f(ST1, ST2, ST16)
STx は，サイズが x[byte] の構造体であることを表している場合の例



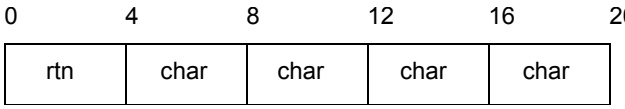
サイズが 4 の倍数でない構造体や共用体の場合，引数同士の間にパディング（図のグレー部分）ができ，この部分は内容が不定となります。

例 2. 関数原型 : f(char, long, ...)
可変個数の実引数を受け取る場合の例



例の「これ以降は可変個数分の領域」では，実引数で設定する個数分だけメモリを消費します。

例 3. 関数原型 : ST4 f(char, char, char, char)



rtn で，ST4 の戻り値を書き込む場所のアドレスを渡します。

(b) 作成したメモリ・イメージの先頭 4 ワード（16 バイト）をレジスタ r6 ～ r9 で，4 ワードに収まらない分をスタックで渡す

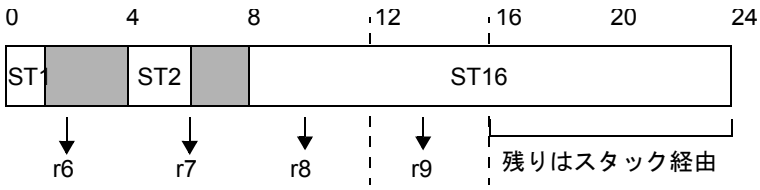
<1> レジスタ渡しの場合，各レジスタ（r6 ～ r9）へは各々ワード単位にロードされます。
バイト単位やハーフワード単位のロードはされません。

<2> スタックで渡す引数は，呼び出し側関数のスタック・フレーム内に設定されます。

<3> スタックで渡す引数をスタックへ設定する場合は，メモリ・イメージの右側から左側にかけてという順番でスタックへ格納されます。そのため，メモリ・イメージの 16 バイト・ オフセット位置のワードデータが，最も 0 に近い位置へ配置されます。

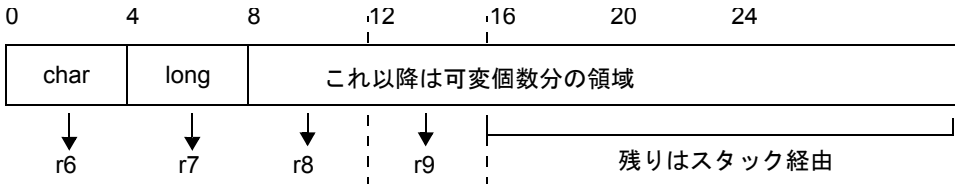
備考 スタックへの配置の方法は，「9.1.4 スタック・フレーム」を参照してください。

例 1. 関数原型 : f(ST1, ST2, ST16)
STx は，サイズが x[byte] の構造体であることを表している場合の例



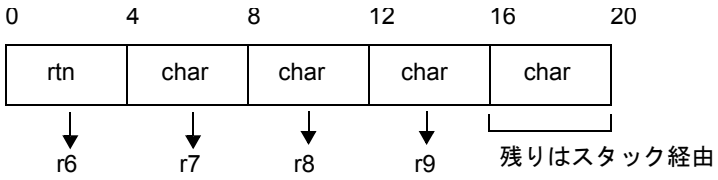
構造体（ここでは “ST16”）が一部分しかレジスタに設定できなくても，かまわずレジスタで渡します。

例 2. 関数原型 : f(char, long, ...)
可変個数の実引数を受け取る場合の例



可変個数であっても，レジスタが使用できる場合はレジスタで受け渡しします。

例 3. 関数原型 : ST4 f(char, char, char, char)



char 型の引数 4 個を渡すだけでも，戻り値によっては，4 番目の引数がスタック渡しとなる場合があります。

(2) 戻り値の受け渡し方法
戻り値の受け渡しの方法には，次の 3 通りがあります。

(a) 4 バイト以下のスカラ型の場合
r10 で戻り値を呼び出し側へ返却します。
4 バイト未満のサイズのスカラ型の場合，4 バイトに拡張したデータを r10 へと設定します。
戻り値の型が , 符号なしであればゼロ拡張 , 符号有りであれば符号拡張します。

(b) 8 バイトのスカラ型の場合
r10，r11 で戻り値を呼び出し側へ返却します。
r10 には下位 32 ビット，r11 には上位 32 ビットを設定します。

(c) 構造体や共用体の場合
戻り値が構造体や共用体の場合，呼び出し側は関数呼び出し時に，引数レジスタ r6 へ戻り値データを書き込む領域のアドレスを設定します。呼び出される側は，パラメータ・レジスタ r6 で指示されるアドレス位置に戻り値を設定し，呼び出し側関数へと返却します。
返却した時点では，呼び出し側の関数にとっては，r6 も r10 も不定（他の関数呼び出し前後で内容が保証されないレジスタと同じ) となります。
構造体や共用体は，サイズによらずすべて同じ返却方法となります。構造体や共用体のデータ自体をレジスタに設定して返却することはしません。

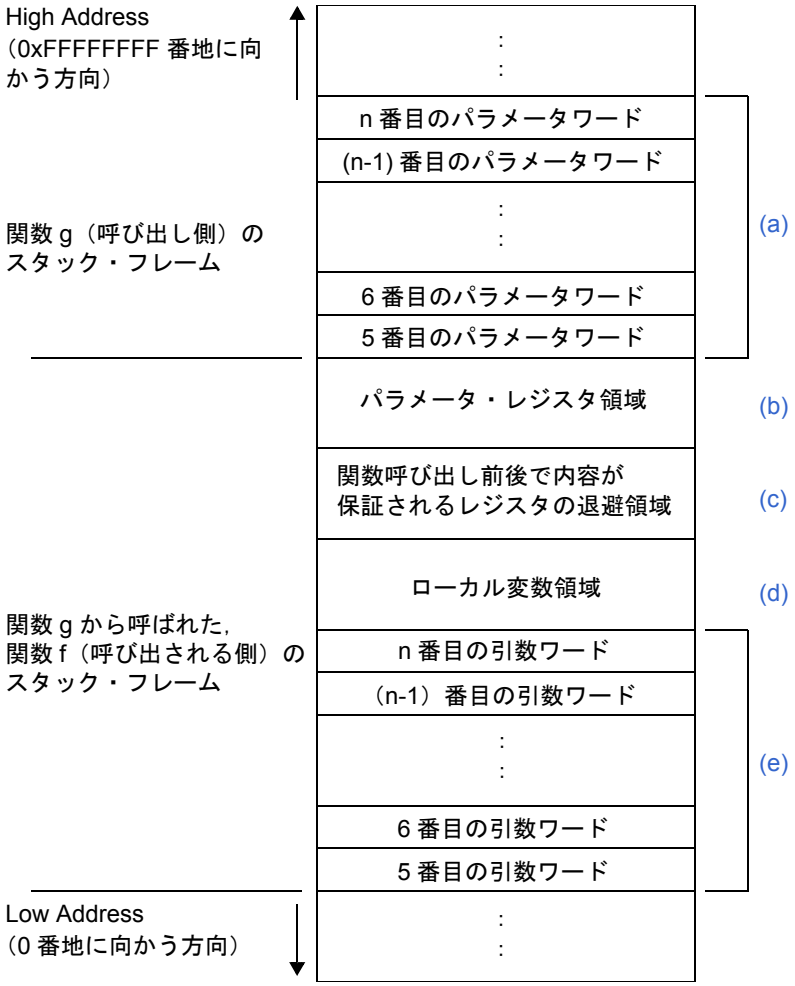
9.1.3 スタック・ポインタが指すアドレス

スタック・ポインタには，4 の倍数のアドレスが設定されます。
スタック・ポインタが指すアドレスは，4 の倍数ですが，スタックに格納するデータをすべて 4 バイト境界に整合する必要があるわけではありません。各データのアライメントに応じて，スタック内に格納される位置が決まります。たとえば，char 型データであれば，データのアライメント数は 1 なので，スタック内でも 1 バイト境界に配置できます。

9.1.4 スタック・フレーム

(1) スタック・フレーム内の構成
関数 f が，関数 g から呼び出された場合の，関数 f から見た双方のスタック・フレームの内容を以下に示します。

図 9.1 スタック・フレームの内容

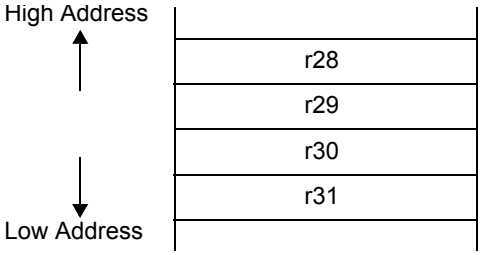


関数 f が参照，または設定できる範囲の領域の内容は以下です。

(a) 5 ～ n 番目のパラメータ・ワード
関数 f が 4 ワード（16 バイト）より大きいパラメータ・サイズを持つ場合に，4 ワードを越えた分のパラメータを設定している領域です。パラメータ・サイズが 4 ワード以下であれば，この領域はサイズ 0 となります。

(b) パラメータ・レジスタ領域
パラメータを受け取るレジスタ（r6 ～ r9）を設定するための領域です。16 バイト固定ではなく，不要であればサイズ 0 となります。
パラメータ・レジスタ領域の詳細は，「(2) パラメータ・レジスタ領域」を参照してください。

(c) 関数呼び出し前後で内容が保証されるレジスタの退避領域
関数 f 内で使用する関数呼び出し前後で内容が保証されるレジスタを退避するための領域です。退避する必要があるレジスタ本数分だけ，この領域のサイズが必要となります。
レジスタの退避と復帰は，基本的に prepare/dispose 命令を使用することになるので，この退避領域へはレジスタ番号昇順にストアされます。
たとえば，r28 ～ r31 を退避する場合は，以下の並び順となります。



(d) ローカル変数領域
ローカル変数として使用するためのスタック領域です。

(e) 5 ～ n 番目の引数ワード

関数 f が他の関数を呼び出す際に，4 ワードより大きいサイズの引数を設定するための領域です。他の関数を呼び出す際に必要となる引数は，関数 f のスタック・フレーム内に領域を確保して設定します。
呼び出しに要する引数サイズが 4 ワード以下の場合，この領域はサイズ 0 となります。

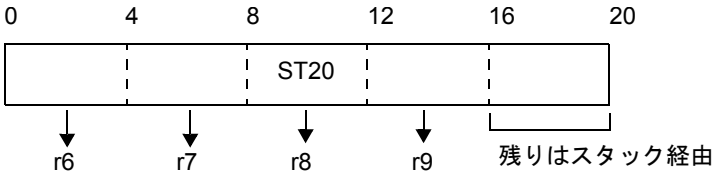
(2) パラメータ・レジスタ領域

パラメータ・レジスタ領域は，パラメータ・サイズが 4 ワード（16 バイト）を越える場合に，必要に応じて確保する領域です。パラメータ・レジスタ（r6 ～ r9）を必要に応じてこの領域にストアするため，この領域のサイズは 0，4，8，12，16 バイトのいずれかです。

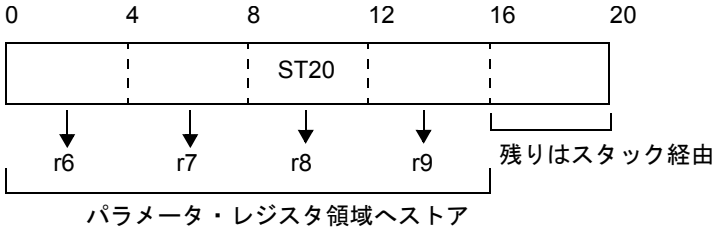
この領域は，パラメータ・レジスタの内容とスタック内のパラメータとを，連続した配置関係で参照する必要がある場合に，パラメータ・レジスタをストアするためのものです。

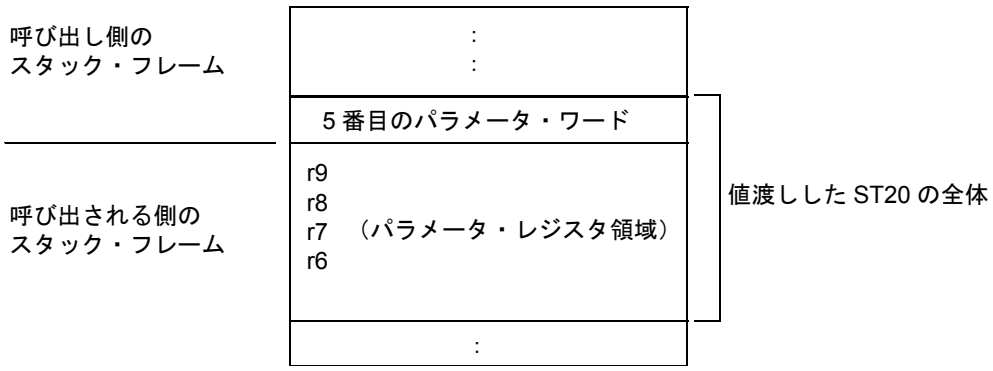
たとえば，サイズが 20 バイトの構造体引数を値渡しする場合，16 バイトは r6 ～ r9 で，残りの 4 バイトはスタック（5 番目のパラメータ・ワード）として渡されます。

例　関数原型 : f(ST20)



値渡しした構造体全体を参照するためには，メモリ上に全体を連続した位置関係で配置する必要がありますが，関数呼び出し直後は，レジスタとメモリに分割された状態になっています。
この場合，呼び出される側の関数でパラメータ・レジスタをスタックへストアすることにより，値渡しした ST20 をメモリ上で参照することができます。

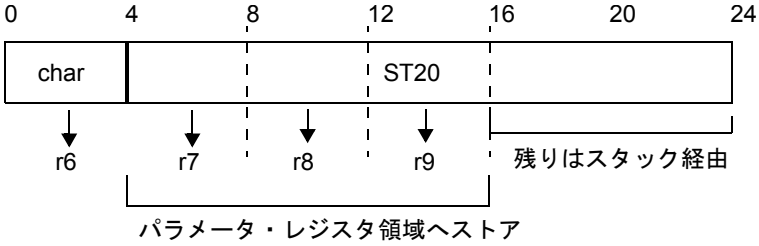




この領域が必要となる具体的なケースは，パラメータが以下の場合です。
これらいずれかに該当しなければ，パラメータ・レジスタをパラメータ・レジスタ領域へストアする必要はないため，パラメータ・レジスタ領域は不要（サイズ 0）となります。

(a) 構造体や共用体が，パラメータ・レジスタとスタックにまたがる場合

例　関数原型 : f(char, ST20)

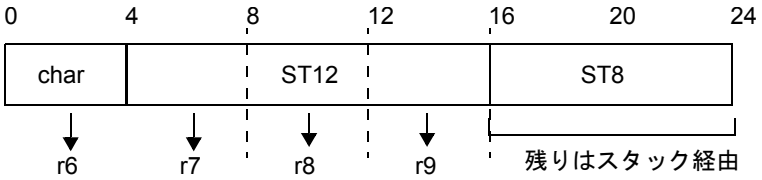


この場合には，r7 ～ r9 をパラメータ・レジスタ領域へストアします。

r6 は，ST20 と連続した位置のメモリ上に配置する必要がないのでストアしません。
したがって，パラメータ・レジスタ領域のサイズは 12 バイトとなります。

なお，構造体や共用体が，パラメータ・レジスタとスタックにまたがっていない場合は，パラメータ・レジスタ領域へのストアは不要になり，パラメータ・レジスタ領域はサイズ 0 バイトとなります。

例　関数原型 : f(char, ST12, ST8)

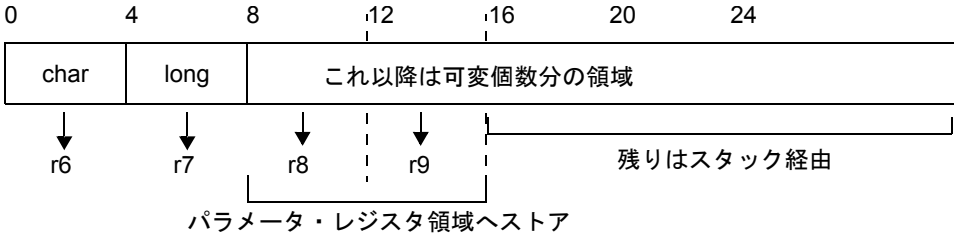


パラメータ・レジスタ領域へストアはありません。

この場合は，ST12 はパラメータ・レジスタ内にすべて収まり，ST8 はパラメータ・レジスタでは渡ってきません。
そのため，パラメータ・レジスタとスタックにはまたがらないので，パラメータ・レジスタ領域はサイズ 0 バイトとなります。
なお，パラメータ・レジスタで全体を受け取った構造体や共用体は，メモリ上に展開する際には，ローカル変数領域に展開されます。

(b) 可変個数の実引数を受け取る場合
可変個数の実引数を受け取る場合には，パラメータ・レジスタ領域へストアする必要があります。

例　関数原型 : f(char, long, ...)



この場合は，可変個数の実引数にあたるパラメータ・レジスタ（r8 と r9）がパラメータ・レジスタ領域へストアされます。
したがって，パラメータ・レジスタ領域のサイズは 8 バイトとなります。

9.2 C 言語からアセンブリ言語ルーチンの呼び出し

C 言語関数からアセンブラ関数を呼び出すときの注意点について説明します。

(1) 識別子について
CC-RH では C ソース内で外部名，たとえば，関数や外部変数が記述された場合，それらの名前をアセンブラへ出力すると，先頭に “_（アンダースコア）” を付けた名前になります。

表 9.1　識別子について

C	アセンブラ
func1 ( )	_func1

アセンブラ命令で関数や外部変数を定義するときは，識別子の先頭に “_” をつけ，C 言語関数から参照するときは “_” を取った形で行ってください。

(2) スタック・フレームに関して
CC-RH は「スタック・ポインタ（SP）が，常にスタック・フレームの最下位アドレスを指している」ことを想定したコードを出力します。そのため，C ソースからアセンブラ関数へ分岐後は，アセンブラ関数内では，SP の指すアドレスよりも下位のアドレス領域は自由に使用することができます。逆に上位のアドレス領域の内容を変更した場合，C 言語関数で使用していた領域を破壊することにつながり，以降の動作を保証できませんので注意が必要です。上記を回避するためには，アセンブラ関数の先頭で SP を変更してからスタックを使用してください。
ただし，その際は呼び出しの前後で SP の値が保持されるようにしてください。

また，アセンブラ関数内でレジスタ変数用レジスタを使用する場合は，アセンブラ関数の呼び出し前後でレジスタ値が保持されるようにしてください（使用前にレジスタ変数用レジスタの値を退避し，使用後は復帰してください）。
レジスタ変数用レジスタは，レジスタ・モードにより異なります。

表 9.2 レジスタ変数用レジスタ

レジスタ・モード	レジスタ変数用レジスタ
22 レジスタ・モード	r25，r26，r27，r28，r29
26 レジスタ・モード	r23，r24，r25，r26，r27，r28，r29
32 レジスタ・モード	r20，r21，r22，r23，r24，r25，r26，r27，r28，r29

(3) C 言語関数への戻り先アドレス
CC-RH は「関数の戻り先アドレスは“リンク・ポインタ lp（r31）”に格納される」ことを想定したコードを生成します。アセンブラ関数へ分岐するとき，lp に関数の戻り先アドレスが格納されているので，C 言語関数へ戻るときは “jmp [lp]” を実行してください。

9.3 アセンブリ言語から C 言語ルーチンの呼び出し

アセンブラ関数から C 言語関数を呼び出すときの注意点について説明します。

(1) スタック・フレームについて
CC-RH は「スタック・ポインタ（SP）が，常にスタック・フレームの最下位アドレスを指している」ことを想定したコードを出力します。そのため，アセンブラ関数から C 言語関数へ分岐する前に，スタック領域中の未使用領域の上位アドレスを指すように SP を設定してください。これは下位アドレスの方向にスタック・フレームが取られるためです。

(2) 作業用レジスタ
CC-RH は C 言語関数呼び出しの前後において，レジスタ変数用レジスタの値は保持しますが，作業用レジスタの値は保持しません。そのため，保持しなくてはならない値を作業用レジスタに割り当てたままにしないでください。
レジスタ変数用レジスタ，作業用レジスタは，レジスタ・モードにより異なります。

表 9.3 レジスタ変数用レジスタ

レジスタ・モード	レジスタ変数用レジスタ
22 レジスタ・モード	r25，r26，r27，r28，r29
26 レジスタ・モード	r23，r24，r25，r26，r27，r28，r29
32 レジスタ・モード	r20，r21，r22，r23，r24，r25，r26，r27，r28，r29

表 9.4 作業用レジスタ

レジスタ・モード	作業用レジスタ
22 レジスタ・モード	r10，r11，r12，r13，r14
26 レジスタ・モード	r10，r11，r12，r13，r14，r15，r16
32 レジスタ・モード	r10，r11，r12，r13，r14，r15，r16，r17，r18，r19

(3) アセンブラ関数への戻り先アドレス
CC-RH は「関数の戻り先アドレスは“リンク・ポインタ lp（r31）”に格納される」ことを想定したコードを生成します。C 言語関数へ分岐するとき，lp に関数の戻り先アドレスを格納する必要あります。
一般的には jarl 命令によって，C 言語関数へ分岐します。

9.4 他言語で定義された変数の参照

アセンブリ言語で定義した変数を C 言語側で参照する方法を以下に示します。

例 C 言語のプログラム

```
extern  char    c;
extern  int     i;

void subf() {
        c = 'A';
        i = 4;
}
```

CC-RH アセンブラでは，次のように行います。

```
        .public _i
        .public _c
        .dseg   SDATA
_i:
        .db4    0x0
_c:
        .db     0x0
```

9.5 汎用レジスタ

以下に，CC-RH における汎用レジスタの用い方を示します。

表 9.5 汎用レジスタの用い方

レジスタ	使用方法
r0	0 の値として演算時に使用 .data/.bss セクション参照のベース・レジスタ
r1	関数呼び出し前後で内容が保証されないレジスタ
r2	関数呼び出し前後で内容が保証されないレジスタ システム予約（OS）（オプションで切り替え）
r3（sp）	スタック・ポインタ
r4（gp）	PID 用グローバル・ポインタ 固定
r5（tp）	定数データ用グローバル・ポインタ 関数呼び出し前後で内容が保証されないレジスタ
r6 ～ r19	関数呼び出し前後で内容が保証されないレジスタ
r20 ～ r29	関数呼び出し前後で内容が保証されるレジスタ
r30（ep）	エレメント・ポインタ 固定，または関数呼び出し前後で内容が保証されるレジスタ（オプションで切り替え）
r31（lp）	リンク・ポインタ 関数呼び出し前後で内容が保証されるレジスタ

10. メッセージ

ここでは，CC-RH が出力するメッセージについて説明します。

10.1 概　　説

ここでは，CC-RH が出力する内部エラー・メッセージ，フェイタル・エラー・メッセージ，アボート・エラー・メッセージ，インフォメーション・メッセージ，ワーニング・メッセージについて説明します。

10.2 出力形式

ここでは，メッセージの出力形式について説明します。
メッセージの出力形式は，次のとおりです。

(1) ファイル名と行番号を含む場合

```
ファイル名 ( 行番号 ) : メッセージ種別 05 メッセージ番号 : メッセージ
```

(2) ファイル名と行番号を含まない場合

```
メッセージ種別 05 メッセージ番号 : メッセージ
```

備考　　下記内容が連続した文字列として出力されます。
メッセージ種別　　:　1 文字の英字
メッセージ番号　　:　5 桁の数値

10.3 メッセージ種別

ここでは，CC-RH が出力するメッセージ種別について説明します。
メッセージ種別（1 文字の英字）は，次のように分類されています。

表 10.1　　メッセージ種別

メッセージ種別	説明
C	内部エラー：処理を中止します。 オブジェクト・コードは生成しません。
E	フェイタル・エラー：一定数以上発生した場合，処理を中止します。 オブジェクト・コードは生成しません。
F	アボート・エラー：処理を中止します。 オブジェクト・コードは生成しません。
M	インフォメーション：処理を続行します。 オブジェクト・コードを生成します。
W	ワーニング：処理を続行します。 オブジェクト・コードを生成します（ユーザが意図したものとは異なる可能性があります）。

10.4 メッセージ番号

ここでは，CC-RH が出力するメッセージ番号について説明します。
CC-RH 実行時のメッセージ番号は，番号 “05” に続けて出力される，5 桁の数値となります。

10.5 メッセージ

ここでは，CC-RH が出力するメッセージについて説明します。

10.5.1 内部エラー

表 10.2 内部エラー

C05*nnnnn*	[メッセージ]	内部エラーが発生しました。
	[メッセージ]	内部エラーが発生しました（*情報*）。
	[対処方法]	特約店，または当社までご連絡ください。
C0510000	[メッセージ]	Internal error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530001	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530002	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530003	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530004	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530005	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0530006	[メッセージ]	Internal Error.
	[対処方法]	特約店，または当社までご連絡ください。
C0560901	[メッセージ]	内部エラーが発生しました。(不正なリンケージステータス (*number*) です。)
	[対処方法]	特約店，または当社までご連絡ください。
C0560903	[メッセージ]	内部エラーが発生しました。(メモリに配置できませんでした。)
	[対処方法]	特約店，または当社までご連絡ください。
C0560904	[メッセージ]	内部エラーが発生しました。(リロケーション・タイプ (*number*) がありません。)
	[対処方法]	特約店，または当社までご連絡ください。
C0560905	[メッセージ]	内部エラーが発生しました。(“*string*” が不正です。)
	[対処方法]	特約店，または当社までご連絡ください。
C0560906	[メッセージ]	内部エラーが発生しました。(未定義の外部シンボル (“*string*”) がみつかりました。)
	[対処方法]	特約店，または当社までご連絡ください。
C0560907	[メッセージ]	内部エラーが発生しました。(GP-symbol 情報が取得できませんでした。)
	[対処方法]	特約店，または当社までご連絡ください。
C0564000	[メッセージ]	Internal error : (" *内部エラー番号* ") " *ファイル 行番号* " / " *コメント* "
	[説明]	最適化リンカの処理中に内部的な問題が発生しました。
	[対処方法]	メッセージ内の内部エラー番号，ファイル，行番号，コメントを添えて，特約店，または当社までご連絡ください。
C0592*xxx*	[メッセージ]	内部エラーが起こりました。
	[対処方法]	特約店，または当社までご連絡ください。

C0592100	［メッセージ］	内部エラーが起こりました。
	［対処方法］	特約店，または当社までご連絡ください。
C0592200	［メッセージ］	内部エラーが発生しました。
	［対処方法］	特約店，または当社までご連絡ください。

10.5.2 フェイタル・エラー

表 10.3 フェイタル・エラー

E0511101	［メッセージ］	" *文字列* " オプションで指定された " *パス名* " はフォルダです。入力ファイルを指定してください。
	［メッセージ］	【CC-RX】 "*path*" specified by the "*character string*" option is a folder. Specify an input file.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された " *パス名* " はフォルダです。入力ファイルを指定してください。
E0511102	［メッセージ］	" *文字列* " オプションで指定されたファイル " *ファイル名* " が見つかりません。
	［メッセージ］	【CC-RX】 The file "*file*" specified by the "*character string*" option is not found
	［説明］	【CC-RX】 " *文字列* " オプションで指定されたファイル " *ファイル名* " が見つかりません。
E0511103	［メッセージ］	" *文字列* " オプションで指定された " *パス名* " はフォルダです。出力ファイルを指定してください。
	［メッセージ］	【CC-RX】 "*path*" specified by the "*character string*" option is a folder. Specify an output file.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された " *パス名* " はフォルダです。出力ファイルを指定してください。
E0511104	［メッセージ］	" *文字列* " オプションで指定された出力先フォルダ " *フォルダ名* " が見つかりません。
	［メッセージ］	【CC-RX】 The output folder "*folder*" specified by the "*character string*" option is not found.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された出力先フォルダ " *フォルダ名* " が見つかりません
E0511107	［メッセージ］	" *文字列* " オプションで指定された " *パス名* " が見つかりません。
	［説明］	" *文字列* " オプションで指定された " *パス名* "（*ファイル名*または *フォルダ名*）が見つかりません。
	［メッセージ］	【CC-RX】 "*path*" specified by the "*character string*" option is not found.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された " *パス名* "（*ファイル名*または *フォルダ名*）が見つかりません。
E0511108	［メッセージ］	" *文字列* " は認識されないオプションです。
	［メッセージ］	【CC-RX】 The "*character string*" option is not recognized.
	［説明］	【CC-RX】 " *文字列* " は認識されないオプションです。
E0511109	［メッセージ］	" *文字列* " オプションに引数は指定できません。
	［メッセージ］	【CC-RX】 The "*character string*" option can not have an argument.
	［説明］	【CC-RX】 " *文字列* " オプションに引数は指定できません。
E0511110	［メッセージ］	" *文字列* " オプションに引数を指定してください。
	［説明］	" *文字列* " オプションは引数が必要です。引数を指定してください。
	［メッセージ］	【CC-RX】 The "*character string*" option requires an argument.
	［説明］	【CC-RX】 " *文字列* " オプションは引数が必要です。引数を指定してください。

E0511111	［メッセージ］	" *文字列* " オプションにパラメータは指定できません。
	［メッセージ］	【CC-RX】 The "*character string*" option can not have a parameter.
	［説明］	【CC-RX】 " *文字列* " オプションにパラメータは指定できません。
E0511112	［メッセージ］	" *文字列* " オプションにパラメータを指定してください。
	［説明］	" *文字列* " オプションはパラメータが必要です。パラメータを指定してください。
	［メッセージ］	【CC-RX】 The "*character string*" option requires a parameter.
	［説明］	【CC-RX】 " *文字列* " オプションはパラメータが必要です。パラメータを指定してください。
E0511113	［メッセージ］	" *文字列* " オプションに指定された引数が不正です。
	［メッセージ］	【CC-RX】 Invalid argument for the "*character string*" option.
	［説明］	【CC-RX】 " *文字列* " オプションに指定された引数が不正です。
E0511114	［メッセージ］	"-O *文字列* " オプションに指定された引数が不正です。
	［メッセージ］	【CC-RX】 Invalid argument for the "-O*character string*" option.
	［説明］	【CC-RX】 "-O *文字列* " オプションに指定された引数が不正です。
E0511115	［メッセージ］	"-O *文字列* " オプションの指定が不正です。
	［メッセージ］	【CC-RX】 The "-O*character string*" option is invalid.
	［説明］	【CC-RX】 "-O *文字列* " オプションの指定が不正です。
E0511116	［メッセージ］	"-O *文字列* " は認識されないオプションです。
	［メッセージ］	【CC-RX】 The "-O*character string*" option is not recognized.
	［説明］	【CC-RX】 "-O *文字列* " は認識されないオプションです。
E0511117	［メッセージ］	" *文字列* " オプションに指定されたパラメータが不正です。
	［メッセージ］	【CC-RX】 Invalid parameter for the "*character string*" option
	［説明］	【CC-RX】 " *文字列* " オプションに指定されたパラメータが不正です。
E0511118	［メッセージ］	" *文字列* " オプションにはシンボルを指定してください。
	［メッセージ］	【CC-RX】 Symbol is required for the "*character string*" option.
	［説明］	【CC-RX】 " *文字列* " オプションにはシンボルを指定してください。
E0511119	［メッセージ］	"-Xr" オプションで指定されたレジスタ " *レジスタ名* " は、コンパイラ・システムで予約されています。
E0511120	［メッセージ］	" *文字列* " オプションには *値 1* ～ *値 2* を指定してください。
	［説明］	指定されたサイズ・オプションの値が，最小値から最大値の値を越えています。
	［対処方法］	サイズ・オプションの値は，最小値から最大値の間で指定してください。
	［メッセージ］	【CC-RX】 Specify a value (*value1* - *value2*) for the "*character string*" option.
	［説明］	【CC-RX】 指定したオプションのパラメータが指定可能範囲外です。

E0511121	［メッセージ］	"-o" オプションと " *文字列* " オプションを同時に指定したとき、複数ソース・ファイルを入力できません。
	［メッセージ］	【CC-RX】 Multiple source files are not allowed when both the "-o" option and the "*character string*" option are specified.
	［説明］	【CC-RX】 "-o" オプションと " *文字列* " オプションを同時に指定したとき，複数ソース・ファイルを入力できません。
E0511122	［メッセージ］	" *文字列* " オプションにはオブジェクト・ファイルを指定してください。
	［メッセージ］	【CC-RX】 The argument for the "*character string*" option must be an object file.
	［説明］	【CC-RX】 " *文字列* " オプションにはオブジェクト・ファイルを指定してください。
E0511124	［メッセージ］	【CC-RH】 "-Xcommon" オプションを指定してください。
	［メッセージ］	【CC-RH】 "-cpu" オプションを指定してください。
E0511125	［メッセージ］	デバイス・ファイルが見つかりません。
	［メッセージ］	【CC-RX】 Cannot find device file.
	［説明］	【CC-RX】 デバイス・ファイルが見つかりません。
E0511126	［メッセージ］	デバイス・ファイル " *ファイル名* " の読み込みに失敗しました。
	［メッセージ］	【CC-RX】 Device file "*file*" read error.
	［説明］	【CC-RX】 デバイス・ファイル " *ファイル名* " の読み込みに失敗しました。
E0511127	［メッセージ］	サポートしていないデバイスが指定されました。
	［メッセージ］	【CC-RX】 The specified device is not supported.
	［説明］	【CC-RX】 サポートしていないデバイスが指定されました。
E0511129	［メッセージ］	【CC-RH】 コマンド・ファイル " *ファイル名* " が複数回読まれています。
	［メッセージ］	【CC-RX】 Command file "*file*" is read more than once.
	［説明］	【CC-RX】 コマンド・ファイル " *ファイル名* " が複数回読まれています。
	［メッセージ］	【CC-RH】 サブコマンド・ファイル " *ファイル名* " が複数回読まれています。
E0511130	［メッセージ］	【CC-RH】 コマンド・ファイル " *ファイル名* " が読み込めません。
	［メッセージ］	【CC-RX】 Command file "*file*" cannot be read.
	［説明］	【CC-RX】 コマンド・ファイル " *ファイル名* " が読み込めません。
	［メッセージ］	【CC-RH】 サブコマンド・ファイル " *ファイル名* " が読み込めません。
E0511131	［メッセージ］	【CC-RH】 コマンド・ファイル " *ファイル名* " の構文が認識できません。
	［メッセージ］	【CC-RX】 Syntax error in command file "*file*".
	［説明］	【CC-RX】 コマンド・ファイル " *ファイル名* " の構文が認識できません。
	［メッセージ］	【CC-RH】 サブコマンド・ファイル " *ファイル名* " の構文が認識できません。
E0511132	［メッセージ］	テンポラリ・フォルダを作成できません。
	［メッセージ］	【CC-RX】 Failed to create temporary folder.
	［説明］	【CC-RX】 テンポラリ・フォルダを作成できません。

E0511133	［メッセージ］	ソース・ファイルが複数の場合は " *文字列* " オプションにはフォルダを指定してください。
	［メッセージ］	【CC-RX】 The parameter for the "*character string*" option must be a folder when multiple source files are specified.
	［説明］	【CC-RX】 ソース・ファイルが複数の場合は " *文字列* " オプションにはフォルダを指定してください。
E0511134	［メッセージ］	指定された入力ファイル " *ファイル名* " が見つかりません。
	［メッセージ］	【CC-RX】 Input file "*file*" is not found.
	［説明］	【CC-RX】 指定された入力ファイル " *ファイル名* " が見つかりません。
E0511135	［メッセージ］	指定された入力ファイル " *パス名* " はフォルダです。
	［メッセージ］	【CC-RX】 "*path*" specified as an input file is a folder.
	［説明］	【CC-RX】 指定された入力ファイル " *パス名* " はフォルダです。
E0511136	［メッセージ］	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
	［メッセージ］	【CC-RX】 Failed to delete a temporary file "*file*".
	［説明］	【CC-RX】 テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0511137	［メッセージ］	テンポラリ・フォルダ " *フォルダ名* " の削除に失敗しました。
	［メッセージ］	【CC-RX】 Failed to delete a temporary folder "*folder*".
	［説明］	【CC-RX】 テンポラリ・フォルダ " *フォルダ名* " の削除に失敗しました。
E0511138	［メッセージ］	入力ファイル " *ファイル名* " のオープンに失敗しました。
	［メッセージ］	【CC-RX】 Failed to open an input file "*file*".
	［説明］	【CC-RX】 入力ファイル " *ファイル名* " のオープンに失敗しました。
E0511139	［メッセージ］	出力ファイル " *ファイル名* " のオープンに失敗しました。
	［メッセージ］	【CC-RX】 Failed to open an output file "*file*".
	［説明］	【CC-RX】 出力ファイル " *ファイル名* " のオープンに失敗しました。
E0511140	［メッセージ］	入力ファイル " *ファイル名* " のクローズに失敗しました。
	［メッセージ］	【CC-RX】 Failed to close an input file "*file*".
	［説明］	【CC-RX】 入力ファイル " *ファイル名* " のクローズに失敗しました。
E0511141	［メッセージ］	出力ファイル " *ファイル名* " の書き込みに失敗しました。
	［メッセージ］	【CC-RX】 Failed to write an output file "*file*".
	［説明］	【CC-RX】 出力ファイル " *ファイル名* " の書き込みに失敗しました。
E0511142	［メッセージ］	" *文字列* " オプションには入力ファイルを複数指定できません。
	［メッセージ］	【CC-RX】 Multiple source files are not allowed when the "*character string*" option is specified.
	［説明］	【CC-RX】 " *文字列* " オプションには入力ファイルを複数指定できません。
E0511145	［メッセージ］	" *文字列 1* " オプションで指定された " *文字列 2* " は使用できません。
	［メッセージ］	【CC-RX】 "*character string2*" specified in the "*character string1*" option is not available.
	［説明］	【CC-RX】 " *文字列 1* " オプションで指定された " *文字列 2* " は使用できません。

E0511148	［メッセージ］	ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。
	［対処方法］	ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。異なるファイル名を指定してください。
	［メッセージ］	【CC-RX】 "*file name*" is specified as an output file for the different options.
	［説明］	【CC-RX】 ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。
E0511150	［メッセージ］	" *文字列 1*" オプションと " *文字列 2*" オプションが矛盾しています。
	［メッセージ］	【CC-RX】 The "*character string1*" option and the "*character string2*" option are inconsistent.
	［説明］	【CC-RX】 " *文字列 1*" オプションと " *文字列 2*" オプションが矛盾しています。
E0511152	［メッセージ］	" *文字列 1*" オプションには " *文字列 2*" オプションが必要です。
	［メッセージ］	【CC-RX】 The "*character string1*" option needs the "*character string2*" option.
	［説明］	【CC-RX】 " *文字列 1*" オプションには " *文字列 2*" オプションが必要です。
E0511154	［メッセージ］	*コンパイラ・パッケージ名*を構成するファイル " *ファイル名* " が見つかりません。再インストールしてください。
	［メッセージ］	【CC-RX】 Component file "*file name*" for the CC-RX is not found. Reinstall the CC-RX.
	［説明］	【CC-RX】 CC-RX 構成するファイル " *ファイル名* " が見つかりません。再インストールしてください。
E0511155	［メッセージ］	" *文字列* " オプションには他のオプションが必要です。
	［メッセージ］	【CC-RX】 The "*character string*" option needs other option(s).
	［説明］	【CC-RX】 " *文字列* " オプションにはほかのオプションが必要です。
E0511157	［メッセージ］	このデバイスには、" *文字列 1*" オプションもしくは、" *文字列 2*" オプションが必要です。
	［メッセージ］	【CC-RX】 The "*character string1*" option or the "*character string2*" option must be specified for this device.
	［説明］	【CC-RX】 このデバイスには，" *文字列 1*" オプションもしくは，" *文字列 2*" オプションが必要です。
E0511158	［メッセージ］	このデバイスは、" *文字列* " オプションをサポートしていません。
	［メッセージ］	【CC-RX】 The "*character string*" option is not supported for this device.
	［説明］	【CC-RX】 このデバイスは，" *文字列* " オプションをサポートしていません。
E0511159	［メッセージ］	" *文字列* " オプションが指定されている時はソースファイルは指定できません。
	［メッセージ］	【CC-RX】 When the "*character string*" option is specified, source files cannot be input.
	［説明］	【CC-RX】 " *文字列* " オプションが指定されている時はソース・ファイルは指定できません。
E0511160	［メッセージ］	このデバイスには、" *文字列* " オプションが必要です。
	［メッセージ］	【CC-RX】 The "*character string*" option must be specified for this device.
	［説明］	【CC-RX】 このデバイスには，" *文字列* " オプションが必要です。

E0511161	［メッセージ］	ファイル " *ファイル名* " の削除に失敗しました。
	［メッセージ］	【CC-RX】 Failed to delete a file "*file*".
	［説明］	【CC-RX】 ファイル " *ファイル名* " の削除に失敗しました。
E0511165	［メッセージ］	Lacking cpu specification.
	［説明］	CPU の指定がされていません。
	［対処方法］	cpu オプション，または環境変数 CPU_RX で CPU を指定してください。
E0511167	［メッセージ］	Illegal section naming.
	［説明］	セクションの命名に誤りがあります。用途の異なるセクションに同じ名前を付けています。
E0511173	［メッセージ］	Failed to access a temporary file
	［説明］	テンポラリ・ファイルの読み書きに失敗しました。
E0511175	［メッセージ］	Neither isa nor cpu is specified.
E0511176	［メッセージ］	Both "-isa" option and "-cpu" option are specified.
E0511200	［メッセージ］	Internal error(*error-information*).
	［説明］	内部エラーが発生しました (*内容*)。
E0512001	［メッセージ］	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
	［メッセージ］	【CC-RX】 Failed to delete a temporary file "*file-name*".
	［説明］	【CC-RX】 テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0520001	［メッセージ］	ファイルの最終行が改行で終了していません。
	［対処方法］	ファイルの最終行が改行で終了していません。改行を追加してください。
E0520002	［メッセージ］	ファイルの最終行の最後にバックスラッシュがあります。
	［メッセージ］	ファイルの最終行の最後にバックスラッシュがあります。削除してください。
E0520005	［メッセージ］	ソース・ファイル " *ファイル名* " を開くことが出来ません。
	［メッセージ］	【CC-RX】 Could not open source file "*file*".
	［説明］	【CC-RX】 ソース・ファイル " *ファイル名* " を開くことができません。
E0520006	［メッセージ］	ファイルの最後までコメントが閉じられていません。
	［対処方法］	ファイルの最後までコメントが閉じられていません。閉じ忘れているコメントがないか確認してください。
	［メッセージ］	【CC-RX】 Comment unclosed at end of file.
	［説明］	【CC-RX】 ファイルの最後までコメントが閉じられていません
	［対処方法］	【CC-RX】 閉じ忘れているコメントがないか確認してください。
E0520007	［メッセージ］	不明なトークンがあります。
	［対処方法］	不明なトークンがあります。該当箇所を確認してください。
	［メッセージ］	【CC-RX】 Unrecognized token.
	［説明］	【CC-RX】 不明なトークンがあります。
	［対処方法］	【CC-RX】 該当箇所を確認してください。

E0520008	［メッセージ］	クォーテーションを閉じられていません。
	［対処方法］	文字列のクォーテーションが閉じられていません。閉じ忘れているクォーテーションがないか確認してください。
	［メッセージ］	【CC-RX】 Missing closing quote.
	［説明］	【CC-RX】 文字列のクォーテーションが閉じられていません。
	［対処方法］	【CC-RX】 閉じ忘れているクォーテーションがないか確認してください。
E0520010	［メッセージ］	"#" はここには書けません。
	［説明］	"#" が正しくない位置に記述されています。
	［メッセージ］	【CC-RX】 # not expected here.
	［説明］	【CC-RX】 "#" はここには書けません。"#" が正しくない位置に記述されています。
E0520011	［メッセージ］	不明な前処理指令があります。
	［メッセージ］	【CC-RX】 Unrecognized preprocessing directive.
	［説明］	【CC-RX】 不明な前処理指令があります。
E0520012	［メッセージ］	前に構文エラーがあるため、ここより文法の解析を再開します。
	［メッセージ］	【CC-RX】 Parsing restarts here after previous syntax error
	［説明］	【CC-RX】 前に構文エラーがあるため，ここより文法の解析を再開します。
E0520013	［メッセージ］	ファイル名がありません。
	［メッセージ］	【CC-RX】 Expected a file name
	［説明］	【CC-RX】 ファイル名がありません。
E0520014	［メッセージ］	前処理指令の後に不正な文字があります。
	［メッセージ］	【CC-RX】 Extra text after expected end of preprocessing directive.
	［説明］	【CC-RX】 前処理指令の後に不正な文字があります。
E0520017	［メッセージ］	"]" がありません。
	［メッセージ］	【CC-RX】 Expected a "]".
	［説明］	【CC-RX】 "]" がありません。
E0520018	［メッセージ］	")" がありません。
	［メッセージ］	【CC-RX】 Expected a ")".
	［説明］	【CC-RX】 ")" がありません。
E0520019	［メッセージ］	数値の後に不正な文字があります。
	［メッセージ］	【CC-RX】 Extra text after expected end of number.
	［説明］	【CC-RX】 数値の後に不正な文字があります。
E0520020	［メッセージ］	識別子 " *文字列* " は定義されていません。
	［メッセージ］	【CC-RX】 Identifier "*character string*" is undefined.
	［説明］	【CC-RX】 識別子 " *文字列* " は定義されていません
E0520022	［メッセージ］	不正な 16 進数です。
	［メッセージ］	【CC-RX】 Invalid hexadecimal number.
	［説明］	［メッセージ］

E0520023	［メッセージ］	定数の値が大きすぎます。
	［メッセージ］	【CC-RX】 Integer constant is too large.
	［説明］	【CC-RX】 定数の値が大きすぎます。
E0520024	［メッセージ］	不正な 8 進数です。
	［説明］	不正な 8 進数です。8 進数に ‘8’，‘9’ は記述できません。
	［メッセージ］	【CC-RX】 Invalid octal digit.
	［説明］	【CC-RX】 不正な 8 進数です。8 進数に ‘8’，‘9’ は記述できません。
E0520025	［メッセージ］	引用文字列は少なくとも 1 文字を含まなければなりません。
	［メッセージ］	【CC-RX】 Quoted string should contain at least one character.
	［説明］	【CC-RX】 引用文字列は少なくとも 1 文字を含まなければなりません。
E0520026	［メッセージ］	Too many characters in character constant.
	［説明］	文字定数中の文字が多すぎます。
E0520027	［メッセージ］	char 型の値が範囲を超えています。
	［メッセージ］	【CC-RX】 Character value is out of range.
	［説明］	【CC-RX】 char 型の値が範囲を越えています。
E0520028	［メッセージ］	式は定数値を持つ必要があります。
	［メッセージ］	【CC-RX】 Expression must have a constant value.
	［説明］	【CC-RX】 式は定数値を持つ必要があります。
E0520029	［メッセージ］	式がありません。
	［メッセージ］	【CC-RX】 Expected an expression.
	［説明］	【CC-RX】 式がありません。
E0520030	［メッセージ］	浮動小数点数定数値が範囲を超えています。
	［メッセージ］	【CC-RX】 Floating constant is out of range.
	［説明］	【CC-RX】 浮動小数点数定数値が範囲を越えています。
E0520031	［メッセージ］	式は整数型を持つ必要があります。
	［メッセージ］	【CC-RX】 Expression must have integral type.
	［説明］	【CC-RX】 式は整数型を持つ必要があります。
E0520032	［メッセージ］	式は算術型を持つ必要があります。
	［メッセージ］	【CC-RX】 Expression must have arithmetic type.
	［説明］	【CC-RX】 式は算術型を持つ必要があります。
E0520033	［メッセージ］	行番号がありません。
	［説明］	#line の後の行番号がありません。
	［メッセージ］	【CC-RX】 Expected a line number.
	［説明］	【CC-RX】 #line の後の行番号がありません。

E0520034	［メッセージ］	不正な行番号です。
	［説明］	#line の後の行番号が不正です。
	［メッセージ］	【CC-RX】 Invalid line number.
	［説明］	【CC-RX】 #line の後の行番号が不正です。
E0520036	［メッセージ］	この前処理指令のための #if がありません。
	［メッセージ］	【CC-RX】 The #if for this directive is missing.
	［説明］	【CC-RX】 この前処理指令のための #if がありません。
E0520037	［メッセージ］	この前処理指令のための #endif がありません。
	［メッセージ］	【CC-RX】 The #endif for this directive is missing.
	［説明］	【CC-RX】 この前処理指令のための #endif がありません。
E0520038	［メッセージ］	この前処理指令は許可されていません -- #else はすでにあります。
	［説明］	#else が複数記述されているため，このディレクティブは不正です。
	［メッセージ］	【CC-RX】 Directive is not allowed -- an #else has already appeared.
	［説明］	【CC-RX】 #else が複数記述されているため，このディレクティブは不正です。
E0520039	［メッセージ］	Division by zero.
	［説明］	0 で除算を行いました。
E0520040	［メッセージ］	識別子がありません。
	［メッセージ］	【CC-RX】 Expected an identifier.
	［説明］	【CC-RX】 識別子がありません。
E0520041	［メッセージ］	式は算術型かポインタ型を持つ必要があります。
	［メッセージ］	【CC-RX】 Expression must have arithmetic or pointer type.
	［説明］	【CC-RX】 式は算術型かポインタ型を持つ必要があります。
E0520042	［メッセージ］	オペランドの型が適合しません (" *型 1*" と " *型 2*")。
	［メッセージ］	【CC-RX】 Operand types are incompatible (" *型 1*" and " *型 2*").
	［説明］	【CC-RX】 オペランドの型が適合しません (" *型 1*" と " *型 2*")。
E0520044	［メッセージ］	式はポインタ型を持つ必要があります。
	［メッセージ］	【CC-RX】 Expression must have pointer type.
	［説明］	【CC-RX】 式はポインタ型を持つ必要があります。
E0520045	［メッセージ］	既定義名に対して #undef を使用できません。
	［メッセージ］	【CC-RX】 #undef may not be used on this predefined name.
	［説明］	【CC-RX】 既定義名に対して #undef を使用できません。
E0520046	［メッセージ］	" *マクロ名* " を再定義することはできません。
	［説明］	" *マクロ名* " は既定義マクロです。再定義することはできません。
	［メッセージ］	【CC-RX】 symbol is predefined; attempted redefinition ignored
	［説明］	【CC-RX】 " *マクロ名* " は既定義マクロです。再定義することはできません。

E0520047	[メッセージ]	マクロ " *マクロ名* " の適合しない再定義があります (宣言位置 行行番号)。
	[説明]	マクロ " *マクロ名* " の再定義が，行番号行での定義と適合しません。
	[メッセージ]	【CC-RX】 Incompatible redefinition of macro "*macro*".
	[説明]	【CC-RX】 マクロ "*macro*" の再定義が，行番号行での定義と適合しません。
E0520049	[メッセージ]	マクロの引数名が重複しています。
	[メッセージ]	【CC-RX】 Duplicate macro parameter name.
	[説明]	【CC-RX】 マクロの引数名が重複しています。
E0520050	[メッセージ]	マクロ定義の最初を "##" とすることはできません。
	[メッセージ]	【CC-RX】 ## may not be first in a macro definition.
	[説明]	【CC-RX】 マクロ定義の最初を "##" とすることはできません。
E0520051	[メッセージ]	マクロ定義の最後を "##" とすることはできません。
	[メッセージ]	【CC-RX】 ## may not be last in a macro definition.
	[説明]	【CC-RX】 マクロ定義の最後を "##" とすることはできません。
E0520052	[メッセージ]	マクロの引数名がありません。
	[メッセージ]	【CC-RX】 Expected a macro parameter name.
	[説明]	【CC-RX】 マクロの引数名がありません。
E0520053	[メッセージ]	":" がありません。
	[メッセージ]	【CC-RX】 Expected a ":".
	[説明]	【CC-RX】 ":" がありません。
E0520054	[メッセージ]	マクロに対する引数が足りません。
	[メッセージ]	【CC-RX】 Too few arguments in macro invocation.
	[説明]	【CC-RX】 マクロに対する引数が足りません。
E0520055	[メッセージ]	マクロに対する引数が多すぎます。
	[メッセージ]	【CC-RX】 Too many arguments in macro invocation.
	[説明]	【CC-RX】 マクロに対する引数が多すぎます。
E0520056	[メッセージ]	sizeof のオペランドに関数は書けません。
	[メッセージ]	【CC-RX】 Operand of sizeof may not be a function.
	[説明]	【CC-RX】 sizeof のオペランドに関数は書けません。
E0520057	[メッセージ]	この演算子は定数式では使用できません。
	[メッセージ]	【CC-RX】 This operator is not allowed in a constant expression.
	[説明]	【CC-RX】 この演算子は定数式では使用できません。
E0520058	[メッセージ]	この演算子はプリプロセッサ用の式には使用できません。
	[メッセージ]	【CC-RX】 This operator is not allowed in a preprocessing expression.
	[説明]	【CC-RX】 この演算子はプリプロセッサ用の式には使用できません。

E0520059	［メッセージ］	定数式の中で関数を呼び出すことはできません。
	［メッセージ］	【CC-RX】 Function call is not allowed in a constant expression.
	［説明］	【CC-RX】 定数式の中で関数を呼び出すことはできません。
E0520060	［メッセージ］	この演算子は整数型定数式には使用できません。
	［メッセージ］	【CC-RX】 This operator is not allowed in an integral constant expression.
	［説明］	【CC-RX】 この演算子は整数型定数式には使用できません。
E0520061	［メッセージ］	整数演算の結果が範囲を超えました。
	［メッセージ］	【CC-RX】 Integer operation result is out of range.
	［説明］	【CC-RX】 整数演算の結果が範囲を越えました。
E0520062	［メッセージ］	シフト数が負数です。
	［メッセージ］	【CC-RX】 Shift count is negative.
	［説明］	【CC-RX】 シフト数が負数です。
E0520063	［メッセージ］	シフト数が多すぎます。
	［メッセージ］	【CC-RX】 Shift count is too large.
	［説明］	【CC-RX】 シフト数が多すぎます。
E0520064	［メッセージ］	宣言は何も宣言できません。
	［メッセージ］	【CC-RX】 Declaration does not declare anything.
	［説明］	【CC-RX】 宣言は何も宣言できません。
E0520065	［メッセージ］	";" がありません。
	［メッセージ］	【CC-RX】 Expected a ";".
	［説明］	【CC-RX】 ";" がありません。
E0520066	［メッセージ］	enum の値が "int" の範囲を越えています。
	［メッセージ］	【CC-RX】 Enumeration value is out of "int" range.
	［説明］	【CC-RX】 enum の値が "int" の範囲を越えています。
E0520067	［メッセージ］	"}" がありません。
	［メッセージ］	【CC-RX】 Expected a "}".
	［説明］	【CC-RX】 "}" がありません。
E0520070	［メッセージ］	不完全型は許されていません。
	［メッセージ］	【CC-RX】 Incomplete type is not allowed.
	［説明］	【CC-RX】 不完全型は許されていません。
E0520071	［メッセージ］	sizeof のオペランドにビット・フィールドは指定できません。
	［メッセージ］	【CC-RX】 Operand of sizeof may not be a bit field.
	［説明］	【CC-RX】 sizeof のオペランドにビット・フィールドは指定できません。
E0520075	［メッセージ］	"*" 演算子のオペランドはポインタ型である必要があります。
	［メッセージ］	【CC-RX】 Operand of "*" must be a pointer.
	［説明］	【CC-RX】 "*" 演算子のオペランドはポインタ型である必要があります。

E0520077	［メッセージ］	宣言に記憶域クラスまたは型指定子がありません。
	［メッセージ］	【CC-RX】 This declaration has no storage class or type specifier.
	［説明］	【CC-RX】 宣言に記憶域クラスまたは型指定子がありません。
E0520078	［メッセージ］	引数宣言に初期化子は書けません。
	［メッセージ］	【CC-RX】 A parameter declaration may not have an initializer.
	［説明］	【CC-RX】 引数宣言に初期化子は書けません。
E0520079	［メッセージ］	型指定子がありません。
	［メッセージ］	【CC-RX】 Expected a type specifier.
	［説明］	【CC-RX】 型指定子がありません。
E0520080	［メッセージ］	記憶域クラスはここでは指定できません。
	［メッセージ］	【CC-RX】 A storage class may not be specified here.
	［説明］	【CC-RX】 記憶域クラスはここでは指定できません。
E0520081	［メッセージ］	複数の記憶域クラスが指定されました。
	［説明］	複数の記憶域クラスが指定されました。記憶域クラスは 1 つしか指定できません。
	［メッセージ］	【CC-RX】 More than one storage class may not be specified.
	［説明］	【CC-RX】 複数の記憶域クラスが指定されました。記憶域クラスは 1 つしか指定できません。
E0520083	［メッセージ］	型修飾子が複数回指定されました。
	［説明］	型修飾子が複数回指定されました。型修飾子は 2 回以上指定できません。
	［メッセージ］	【CC-RX】 Type qualifier specified more than once.
	［説明］	【CC-RX】 型修飾子が複数回指定されました。型修飾子は 2 回以上指定できません。
E0520084	［メッセージ］	不正な型指定子の組み合わせです。
	［メッセージ］	【CC-RX】 Invalid combination of type specifiers.
	［説明］	【CC-RX】 不正な型指定子の組み合わせです。
E0520085	［メッセージ］	引数に対する記憶域クラスが不正です。
	［メッセージ］	【CC-RX】 Invalid storage class for a parameter.
	［説明］	【CC-RX】 引数に対する記憶域クラスが不正です。
E0520086	［メッセージ］	関数に対する記憶域クラスが不正です。
	［メッセージ］	【CC-RX】 Invalid storage class for a function.
	［説明］	【CC-RX】 関数に対する記憶域クラスが不正です。
E0520087	［メッセージ］	型指定子はここでは使用できません。
	［メッセージ］	【CC-RX】 A type specifier may not be used here.
	［説明］	【CC-RX】 型指定子はここでは使用できません。
E0520088	［メッセージ］	関数の配列は許されていません。
	［メッセージ］	【CC-RX】 Array of functions is not allowed.
	［説明］	【CC-RX】 関数の配列は許されていません。

E0520089	［メッセージ］	void 型の配列は許されていません。
	［メッセージ］	【CC-RX】 Array of void is not allowed.
	［説明］	【CC-RX】 void 型の配列は許されていません。
E0520090	［メッセージ］	関数を返す関数は許されていません。
	［メッセージ］	【CC-RX】 Function returning function is not allowed.
	［説明］	【CC-RX】 関数を返す関数は許されていません。
E0520091	［メッセージ］	配列を返す関数は許されていません。
	［メッセージ］	【CC-RX】 Function returning array is not allowed.
	［説明］	【CC-RX】 配列を返す関数は許されていません。
E0520092	［メッセージ］	引数の識別子リストは関数定義でのみ利用できます。
	［メッセージ］	【CC-RX】 Identifier-list parameters may only be used in a function definition.
	［説明］	【CC-RX】 引数の識別子リストは関数定義でのみ利用できます。
E0520093	［メッセージ］	関数型は typedef に指定できません。
	［メッセージ］	【CC-RX】 Function type may not come from a typedef.
	［説明］	【CC-RX】 関数型は typedef に指定できません。
E0520094	［メッセージ］	配列のサイズは正の整数でなければなりません。
	［メッセージ］	【CC-RX】 The size of an array must be greater than zero.
	［説明］	【CC-RX】 配列のサイズは正の整数でなければなりません。
E0520095	［メッセージ］	配列が大きすぎます。
	［メッセージ］	【CC-RX】 Array is too large.
	［説明］	【CC-RX】 配列が大きすぎます。
E0520096	［メッセージ］	コンパイル単位は少なくとも 1 つの宣言を含まなければなりません。
	［メッセージ］	【CC-RX】 A translation unit must contain at least one declaration.
	［説明］	【CC-RX】 コンパイル単位は少なくとも 1 つの宣言を含まなければなりません。
E0520097	［メッセージ］	この型の値は関数返却値にできません。
	［メッセージ］	【CC-RX】 A function may not return a value of this type.
	［説明］	【CC-RX】 この型の値は関数返却値にできません。
E0520098	［メッセージ］	この型の配列は許されていません。
	［メッセージ］	【CC-RX】 An array may not have elements of this type.
	［説明］	【CC-RX】 この型の配列は許されていません。
E0520099	［メッセージ］	ここでの宣言は引数宣言でなければなりません。
	［メッセージ］	【CC-RX】 A declaration here must declare a parameter.
	［説明］	【CC-RX】 ここでの宣言は引数宣言でなければなりません。
E0520100	［メッセージ］	引数名が重複しています。
	［メッセージ］	【CC-RX】 Duplicate parameter name.
	［説明］	【CC-RX】 引数名が重複しています。

E0520101	［メッセージ］	" *シンボル名* " はすでにこのスコープで宣言されています。
	［メッセージ］	【CC-RX】 " *シンボル名* " has already been declared in the current scope.
	［説明］	【CC-RX】 " *シンボル名* " はすでにこのスコープで宣言されています。
E0520102	［メッセージ］	列挙型の前方宣言は標準でありません。
	［メッセージ］	【CC-RX】 Forward declaration of enum type is nonstandard.
	［説明］	【CC-RX】 列挙型の前方宣言は標準でありません。
E0520103	［メッセージ］	Class is too large.
	［説明］	クラスが大きすぎます。
E0520104	［メッセージ］	構造体または共用体が大きすぎます。
	［メッセージ］	【CC-RX】 Struct or union is too large.
	［説明］	【CC-RX】 構造体または共用体が大きすぎます。
E0520105	［メッセージ］	ビット・フィールドのサイズが不正です。
	［メッセージ］	【CC-RX】 Invalid size for bit field.
	［説明］	【CC-RX】 ビット・フィールドのサイズが不正です。
E0520106	［メッセージ］	ビット・フィールドの型が不正です。
	［メッセージ］	【CC-RX】 Invalid type for a bit field.
	［説明］	【CC-RX】 ビット・フィールドの型が不正です。
E0520107	［メッセージ］	サイズ 0 のビット・フィールドは名前を持てません。
	［メッセージ］	【CC-RX】 Zero-length bit field must be unnamed.
	［説明］	【CC-RX】 サイズ 0 のビット・フィールドは名前を持てません。
E0520109	［メッセージ］	式は関数型または関数ポインタ型でなければなりません。
	［メッセージ］	【CC-RX】 Expression must have (pointer-to-) function type.
	［説明］	【CC-RX】 式は関数型または関数ポインタ型でなければなりません。
E0520110	［メッセージ］	タグ名または定義がありません。
	［メッセージ］	【CC-RX】 Expected either a definition or a tag name.
	［説明］	【CC-RX】 タグ名または定義がありません。
E0520112	［メッセージ］	"while" がありません。
	［メッセージ］	【CC-RX】 Expected "while".
	［説明］	【CC-RX】 "while" がありません。
E0520114	［メッセージ］	タイプ " *シンボル* " は参照されていますが定義されていません。
	［メッセージ］	【CC-RX】 Type "*symbol*" was referenced but not defined.
	［説明］	【CC-RX】 タイプ " *シンボル* " は参照されていますが定義されていません。
E0520115	［メッセージ］	continue 文はループの中でのみ使用できます。
	［メッセージ］	【CC-RX】 A continue statement may only be used within a loop.
	［説明］	【CC-RX】 continue 文はループの中でのみ使用できます。

E0520116	［メッセージ］	break 文はループまたは switch の中でのみ使用できます。
	［メッセージ］	【CC-RX】 A break statement may only be used within a loop or switch.
	［説明］	【CC-RX】 break 文はループまたは switch の中でのみ使用できます。
E0520118	［メッセージ］	void 関数は値を返せません。
	［メッセージ］	【CC-RX】 A void function may not return a value.
	［説明］	【CC-RX】 void 関数は値を返せません。
E0520119	［メッセージ］	" *型名* " 型へのキャストは許されていません。
	［メッセージ］	【CC-RX】 Cast to type "*type*" is not allowed.
	［説明］	【CC-RX】 "*type*" 型へのキャストは許されていません。
E0520120	［メッセージ］	返却値の型が関数の型と合っていません。
	［メッセージ］	【CC-RX】 Return value type does not match the function type.
	［説明］	【CC-RX】 返却値の型が関数の型と合っていません。
E0520121	［メッセージ］	case ラベルは switch の中でのみ使用できます。
	［メッセージ］	【CC-RX】 A case label may only be used within a switch.
	［説明］	【CC-RX】 case ラベルは switch の中でのみ使用できます。
E0520122	［メッセージ］	default ラベルは switch の中でのみ使用できます。
	［メッセージ］	【CC-RX】 A default label may only be used within a switch.
	［説明］	【CC-RX】 default ラベルは switch の中でのみ使用できます。
E0520123	［メッセージ］	この case ラベルの値はすでにこの switch の中で使用されています。
	［メッセージ］	【CC-RX】 case label value has already appeared in this switch.
	［説明］	【CC-RX】 この case ラベルの値はすでにこの switch の中で使用されています。
E0520124	［メッセージ］	default ラベルはすでにこの switch の中で使用されています。
	［メッセージ］	【CC-RX】 default label has already appeared in this switch.
	［説明］	【CC-RX】 default ラベルはすでにこの switch の中で使用されています。
E0520125	［メッセージ］	"(" がありません。
	［メッセージ］	【CC-RX】 Expected a "(".
	［説明］	【CC-RX】 "(" がありません。
E0520126	［メッセージ］	式は左辺値である必要があります。
	［メッセージ］	【CC-RX】 Expression must be an lvalue.
	［説明］	【CC-RX】 式は左辺値である必要があります。
E0520127	［メッセージ］	文がありません。
	［メッセージ］	【CC-RX】 Expected a statement.
	［説明］	【CC-RX】 文がありません。
E0520129	［メッセージ］	ブロック・スコープの関数は extern 記憶域クラスのみ指定できます。
	［メッセージ］	【CC-RX】 A block-scope function may only have extern storage class.
	［説明］	【CC-RX】 ブロック・スコープの関数は extern 記憶域クラスのみ指定できます。

E0520130	［メッセージ］	"{" がありません。
	［メッセージ］	【CC-RX】 Expected a "{".
	［説明］	【CC-RX】 "{" がありません。
E0520131	［メッセージ］	Expression must have pointer-to-class type.
	［説明］	式はクラスへのポインタ型でなければなりません。
E0520132	［メッセージ］	式は構造体か共用体へのポインタでなければなりません。
	［メッセージ］	【CC-RX】 Expression must have pointer-to-struct-or-union type.
	［説明］	【CC-RX】 式は構造体か共用体へのポインタでなければなりません。
E0520133	［メッセージ］	Expected a member name.
	［説明］	メンバ名が不正です。または，ありません。
E0520134	［メッセージ］	フィールド名がありません。
	［メッセージ］	【CC-RX】 Expected a field name.
	［説明］	【CC-RX】 フィールド名がありません。
E0520135	［メッセージ］	*symbol* has no member *member*.
	［説明］	*symbol* はメンバ *member* を持ちません。
E0520136	［メッセージ］	*種別* " *シンボル名* " はフィールド " *フィールド名* " を持ちません。
	［メッセージ］	【CC-RX】 *種別* " *シンボル名* " has no field " *フィールド名* ".
	［説明］	【CC-RX】 *種別* " *シンボル名* " はフィールド " *フィールド名* " を持ちません。
E0520137	［メッセージ］	式は変更可能な左辺値である必要があります。
	［メッセージ］	【CC-RX】 Expression must be a modifiable lvalue.
	［説明］	【CC-RX】 式は変更可能な左辺値である必要があります。
E0520138	［メッセージ］	レジスタ変数に対するアドレス演算子は許されていません。
	［メッセージ］	【CC-RX】 Taking the address of a register variable is not allowed.
	［説明］	【CC-RX】 レジスタ変数に対するアドレス演算子は許されていません。
E0520139	［メッセージ］	ビット・フィールドに対するアドレス演算子は許されていません。
	［メッセージ］	【CC-RX】 Taking the address of a bit field is not allowed.
	［説明］	【CC-RX】 ビット・フィールドに対するアドレス演算子は許されていません。
E0520140	［メッセージ］	関数呼び出しに対する引数が多すぎます。
	［メッセージ］	【CC-RX】 Too many arguments in function call.
	［説明］	【CC-RX】 関数呼び出しに対する引数が多すぎます。
E0520141	［メッセージ］	名前なしでプロトタイプ宣言された引数は関数定義がある場合には許されていません。
	［メッセージ］	【CC-RX】 Unnamed prototyped parameters not allowed when body is present.
	［説明］	【CC-RX】 名前なしでプロトタイプ宣言された引数は関数定義がある場合には許されていません。

E0520142	［メッセージ］	式はオブジェクト型へのポインタである必要があります。
	［メッセージ］	【CC-RX】 Expression must have pointer-to-object type.
	［説明］	【CC-RX】 式はオブジェクト型へのポインタである必要があります。
E0520144	［メッセージ］	型 " *型名 1"* の値は型 " *型名 2*" の実体の初期化には使用できません。
	［メッセージ］	【CC-RX】 A value of type " *型名 1*" cannot be used to initialize an entity of type " *型名 2*".
	［説明］	【CC-RX】 型 " *型名 1"* の値は型 " *型名 2*" の実体の初期化には使用できません。
E0520145	［メッセージ］	*種別* " *シンボル名* " は初期化できません。
	［メッセージ］	【CC-RX】 *Type* "*symbol*" may not be initialized.
	［説明］	【CC-RX】 *Type* "*symbol*" は初期化できません。
E0520146	［メッセージ］	初期化子が多すぎます。
	［メッセージ］	【CC-RX】 Too many initializer values.
	［説明］	【CC-RX】 初期化子が多すぎます。
E0520147	［メッセージ］	宣言は " *宣言* "(宣言位置 行 *行番号*) と整合しません。
	［メッセージ］	【CC-RX】 Declaration is incompatible with "*declaration*".
	［説明］	【CC-RX】 宣言は "*declaration*" と整合しません。
E0520148	［メッセージ］	*種別* " *シンボル名* " はすでに初期化されています。
	［メッセージ］	【CC-RX】 *Tyep* "*symbol*" has already been initialized.
	［説明］	【CC-RX】 *Tyep* "*symbol*" はすでに初期化されています。
E0520149	［メッセージ］	グローバル・スコープの宣言ではこの記憶域クラスを指定できません。
	［メッセージ］	【CC-RX】 A global-scope declaration may not have this storage class.
	［説明］	【CC-RX】 グローバル・スコープの宣言ではこの記憶域クラスを指定できません。
E0520150	［メッセージ］	A type name may not be redeclared as a parameter.
	［説明］	型名は引数として再宣言できません。
E0520151	［メッセージ］	typedef 名は引数として再宣言できません。
	［メッセージ］	【CC-RX】 A typedef name may not be redeclared as a parameter.
	［説明］	【CC-RX】 typedef 名は引数として再宣言できません。
E0520153	［メッセージ］	Expression must have class type.
	［説明］	式はクラス型である必要があります。
E0520154	［メッセージ］	式は構造体または共用体型である必要があります。
	［メッセージ］	【CC-RX】 Expression must have struct or union type.
	［説明］	【CC-RX】 式は構造体または共用体型である必要があります。
E0520157	［メッセージ］	式は整数定数式である必要があります。
	［メッセージ］	【CC-RX】 Expression must be an integral constant expression.
	［説明］	【CC-RX】 式は整数定数式である必要があります。

E0520158	［メッセージ］	式は左辺値か関数指示子である必要があります。
	［メッセージ］	【CC-RX】 Expression must be an lvalue or a function designator.
	［説明］	【CC-RX】 式は左辺値か関数指示子である必要があります。
E0520159	［メッセージ］	宣言は以前の " *宣言* "(宣言位置 行 *行番号*) と整合しません。
	［メッセージ］	【CC-RX】 Declaration is incompatible with previous "*symbol*".
	［説明］	【CC-RX】 宣言は以前の "*symbol*" と整合しません。
E0520160	［メッセージ］	External name conflicts with external name of "*symbol*".
	［説明］	外部名が "*symbol*" と矛盾しています。
E0520165	［メッセージ］	関数呼び出しに引数が足りません。
	［メッセージ］	【CC-RX】 Too few arguments in function call.
	［説明］	【CC-RX】 関数呼び出しに引数が足りません。
E0520166	［メッセージ］	不正な浮動小数点定数です。
	［メッセージ］	【CC-RX】 Invalid floating constant.
	［説明］	【CC-RX】 不正な浮動小数点定数です。
E0520167	［メッセージ］	" *型名 1*" 型の引数は型 " *型名 2*" の引数と整合しません。
	［メッセージ］	【CC-RX】 Argument of type "*type1*" is incompatible with parameter of type "*type2*".
	［説明］	【CC-RX】 "*type1*" 型の引数は型 "*type2*" の引数と整合しません。
E0520168	［メッセージ］	関数型はここでは許されていません。
	［メッセージ］	【CC-RX】 A function type is not allowed here.
	［説明］	【CC-RX】 関数型はここでは許されていません。
E0520169	［メッセージ］	宣言がありません。
	［メッセージ］	【CC-RX】 Expected a declaration.
	［説明］	【CC-RX】 宣言がありません。
E0520171	［メッセージ］	不正な型変換です。
	［メッセージ］	【CC-RX】 Invalid type conversion.
	［説明］	【CC-RX】 不正な型変換です。
E0520172	［メッセージ］	外部または内部リンケージが以前の宣言と整合しません。
E0520173	［メッセージ］	浮動小数点数は要求された整数型に入りません。
	［メッセージ］	【CC-RX】 Floating-point value does not fit in required integral type.
	［説明］	【CC-RX】 浮動小数点数は要求された整数型に入りません。
E0520179	［メッセージ］	"%" の右オペランドが 0 です。
	［メッセージ］	【CC-RX】 Right operand of "%" is zero.
	［説明］	【CC-RX】 "%" の右オペランドが 0 です。
E0520183	［メッセージ］	キャストの型は整数型である必要があります。
	［メッセージ］	【CC-RX】 Type of cast must be integral.
	［説明］	【CC-RX】 キャストの型は整数型である必要があります。

E0520184	［メッセージ］	キャストの型は算術型かポインタ型である必要があります。
	［メッセージ］	【CC-RX】 Type of cast must be arithmetic or pointer.
	［説明］	【CC-RX】 キャストの型は算術型かポインタ型である必要があります。
E0520194	［メッセージ］	アセンブラ文字列がありません。
	［説明］	__asm() 中にアセンブラ文字列がありません。
	［メッセージ］	【CC-RX】 Expected an asm string.
	［説明］	【CC-RX】 __asm() 中にアセンブラ文字列がありません。
E0520195	［メッセージ］	An asm function must be prototyped.
	［説明］	asm 関数はプロトタイプ宣言される必要があります。
E0520196	［メッセージ］	An asm function may not have an ellipsis.
	［説明］	asm 関数に省略記号は使用できません。
E0520220	［メッセージ］	整数値が要求された浮動小数点型に入りません。
	［メッセージ］	【CC-RX】 Integral value does not fit in required floating-point type.
	［説明］	【CC-RX】 整数値が要求された浮動小数点型に入りません。
E0520221	［メッセージ］	浮動小数点数値が要求された浮動小数点型に入りません。
	［メッセージ］	【CC-RX】 Floating-point value does not fit in required floating-point type.
	［説明］	【CC-RX】 浮動小数点数値が要求された浮動小数点型に入りません
E0520222	［メッセージ］	浮動小数点演算の結果が範囲を超えました。
	［メッセージ］	【CC-RX】 Floating-point operation result is out of range.
	［説明］	【CC-RX】 浮動小数点演算の結果が範囲を越えました。
E0520227	［メッセージ］	Macro recursion.
	［説明］	マクロが再帰しています。
E0520228	［メッセージ］	カンマが仕様にあっていません。
	［説明］	最後のカンマは標準ではありません。
	［メッセージ］	【CC-RX】 Trailing comma is nonstandard.
	［説明］	【CC-RX】 最後のカンマは標準ではありません
E0520230	［メッセージ］	ビット・フィールドに対し標準でない型です。
	［メッセージ］	【CC-RX】 Nonstandard type for a bit field.
	［説明］	【CC-RX】 ビット・フィールドに対し標準でない型です。
E0520235	［メッセージ］	変数 " *変数名* " が不完全型で宣言されました。
	［メッセージ］	【CC-RX】 Variable *変数名* was declared with a never-completed type.
	［説明］	【CC-RX】 変数 " *変数名* " が不完全型で宣言されました。
E0520238	［メッセージ］	引数の指定子が不正です。
	［メッセージ］	【CC-RX】 Invalid specifier on a parameter
	［説明］	【CC-RX】 引数の指定子が不正です。

E0520239	[メッセージ]	Invalid specifier outside a class declaration.
	[説明]	クラス宣言外で不正な指定子を使用しています。
E0520240	[メッセージ]	宣言の指示子が重複しています。
	[メッセージ]	【CC-RX】 Duplicate specifier in declaration.
	[説明]	【CC-RX】 宣言の指示子が重複しています。
E0520241	[メッセージ]	A union is not allowed to have a base class.
	[説明]	union 型は基底クラスを持つことはできません。
E0520242	[メッセージ]	Multiple access control specifiers are not allowed.
	[説明]	アクセス指定子が重複して使われています。
E0520243	[メッセージ]	class or struct definition is missing.
	[説明]	class 定義の括弧の対応がとれません。
E0520244	[メッセージ]	Qualified name is not a member of class *型* or its base classes.
	[説明]	限定名がクラスまたは基底クラスのメンバの " *型* " でありません。
E0520245	[メッセージ]	A nonstatic member reference must be relative to a specific object.
	[説明]	非静的メンバの参照がオブジェクトに対応していません。
E0520246	[メッセージ]	A nonstatic data member may not be defined outside its class.
	[説明]	非静的データ・メンバはクラス外で定義できません。
E0520247	[メッセージ]	*種別* " *シンボル名* " はすでに定義されています。
	[メッセージ]	【CC-RX】 "*symbol*" has already been defined.
	[説明]	【CC-RX】 "*symbol*" はすでに定義されています。
E0520248	[メッセージ]	Pointer to reference is not allowed.
	[説明]	リファレンス型へのポインタ型は許されません。
E0520249	[メッセージ]	Reference to reference is not allowed.
	[説明]	リファレンス型へのリファレンス型は許されません。
E0520250	[メッセージ]	Reference to void is not allowed.
	[説明]	void 型へのリファレンス型は許されません。
E0520251	[メッセージ]	Array of reference is not allowed.
	[説明]	リファレンス型の配列は許されません。
E0520252	[メッセージ]	Reference " *名前* " requires an initializer.
	[説明]	リファレンス型の定義 " *名前* " には初期値が必要です。
E0520253	[メッセージ]	"," がありません。
	[メッセージ]	【CC-RX】 Expected a ",".
	[説明]	【CC-RX】 "," がありません。
E0520254	[メッセージ]	型名は許されていません。
	[メッセージ]	【CC-RX】 Type name is not allowed.
	[説明]	【CC-RX】 型名は許されていません。

E0520255	［メッセージ］	型定義は許されていません。
	［メッセージ］	【CC-RX】 Type definition is not allowed.
	［説明］	【CC-RX】 型定義は許されていません。
E0520256	［メッセージ］	型名 " *型名* " が不正に再宣言されています。
	［メッセージ］	【CC-RX】 Invalid redeclaration of type name "*symbol*"
	［説明］	【CC-RX】 型名 "*symbol*" が不正に再宣言されています。
E0520257	［メッセージ］	const *種別* " *シンボル名* " は初期化子が必要です。
E0520258	［メッセージ］	"this" may only be used inside a nonstatic member function.
	［説明］	"this" が非静的メンバ関数以外で使われています。
E0520259	［メッセージ］	定数値が不明です。
	［メッセージ］	【CC-RX】 Constant value is not known.
	［説明］	【CC-RX】 定数値が不明です。
E0520260	［メッセージ］	明示的な型がありません。"int" として扱います。
	［メッセージ］	【CC-RX】 Explicit type is missing ("int" assumed).
	［説明］	【CC-RX】 明示的な型がありません。"int" として扱います。
E0520262	［メッセージ］	Not a class or struct name.
	［説明］	基底クラスで指定されたクラスまたは構造体がありません。
E0520263	［メッセージ］	Duplicate base class name.
	［説明］	基底クラスを二重に指定しています。
E0520264	［メッセージ］	Invalid base class.
	［説明］	基底クラスが不正です。
E0520265	［メッセージ］	" *名前* " is inaccessible.
	［説明］	" *名前* " をアクセスすることはできません。
E0520266	［メッセージ］	" *名前* " is ambiguous.
	［説明］	指定された " *名前* " があいまいです。
E0520267	［メッセージ］	Old-style parameter list (anachronism).
	［説明］	古い仕様の引数リストです。
E0520268	［メッセージ］	ブロック内で実行文の後に宣言を置けません。
	［メッセージ］	【CC-RX】 Declaration may not appear after executable statement in block.
	［説明］	【CC-RX】 ブロック内で実行文の後に宣言を置けません。
E0520269	［メッセージ］	Conversion to inaccessible base class " *型* " is not allowed.
	［説明］	参照不可能な基底クラス " *型* " に変換できません。
E0520274	［メッセージ］	不適切に終了したマクロの呼び出しがあります。
	［メッセージ］	【CC-RX】 Improperly terminated macro invocation.
	［説明］	【CC-RX】 不適切に終了したマクロの呼び出しがあります。

E0520276	[メッセージ]	Name followed by "::" must be a class or namespace name.
	[説明]	"::" に続く名前は class 名または namespace 名でなければなりません。
E0520277	[メッセージ]	Invalid friend declaration.
	[説明]	フレンド宣言の指定が正しくありません。
E0520278	[メッセージ]	A constructor or destructor may not return a value.
	[説明]	コンストラクタやデストラクタはリターン値を持てません。
E0520279	[メッセージ]	Invalid destructor declaration.
	[説明]	デストラクタの宣言が正しくありません。
E0520280	[メッセージ]	Declaration of a member with the same name as its class.
	[説明]	クラス名と同じ名前のメンバ名を宣言しています。
E0520281	[メッセージ]	Global-scope qualifier (leading "::") is not allowed.
	[説明]	グローバルなスコープ決定演算子は許されません。
E0520282	[メッセージ]	The global scope has no *xxx*.
	[説明]	グローバルスコープは *xxx* を持ちません。
E0520283	[メッセージ]	Qualified name is not allowed.
	[説明]	修飾名は許されていません。
E0520284	[メッセージ]	NULL reference is not allowed.
	[説明]	NULL へのリファレンスは許されません。指定された通りに式を評価します。
E0520285	[メッセージ]	Initialization with "{...}" is not allowed for object of type "*type*".
	[説明]	{...} を伴った初期化は "*type*" 型のオブジェクトには許されていません。
E0520286	[メッセージ]	Base class " *型* " is ambiguous.
	[説明]	基底クラスの型があいまいです。
E0520287	[メッセージ]	Derived class *型 1* contains more than one instance of class *型 2*.
	[説明]	派生型が複数の同一クラス " *型* " を含みます。
E0520288	[メッセージ]	Cannot convert pointer to base class *型 2* to pointer to derived class *型 1* -- base class is virtual.
	[説明]	仮想基底クラス " *型 1*" のポインタ型を派生クラス " *型 2*" のポインタ型に変換することはできません。
E0520289	[メッセージ]	No instance of constructor *名前* matches the argument list.
	[説明]	コンストラクタ " *名前* " の引数が一致しません。
E0520290	[メッセージ]	Copy constructor for class *型* is ambiguous.
	[説明]	クラス " *型* " のコピーコンストラクタがあいまいです。
E0520291	[メッセージ]	No default constructor exists for class *型* .
	[説明]	クラス " *型* " のデフォルトコンストラクタは存在しません。
E0520292	[メッセージ]	*名前* is not a nonstatic data member or base class of class *型* .
	[説明]	" *名前* " が非静的データ・メンバまたは基底クラス " *型* " でありません。

E0520293	[メッセージ]	Indirect nonvirtual base class is not allowed.
	[説明]	仮想でない間接基底クラスは許されません。
E0520294	[メッセージ]	Invalid union member -- class *型* has a disallowed member function.
	[説明]	union メンバに指定できないクラス " *型* " のメンバ関数があります。
E0520296	[メッセージ]	Invalid use of non-lvalue array.
	[説明]	左辺値でない配列の不正な利用です。
E0520297	[メッセージ]	Expected an operator.
	[説明]	演算子が必要です。
E0520298	[メッセージ]	Inherited member is not allowed.
	[説明]	継承されたメンバを使用することはできません。
E0520299	[メッセージ]	Cannot determine which instance of *名前* is intended.
	[説明]	オーバーロード関数の " *名前* " を決定できません。
E0520300	[メッセージ]	A pointer to a bound function may only be used to call the function.
	[説明]	メンバ関数へのポインタを関数呼び出し以外に使用しています。
E0520301	[メッセージ]	*typedef* 名はすでに同じ型で宣言されています。
	[メッセージ]	【CC-RX】 *typedef name* has already been declared (with same type).
	[説明]	【CC-RX】 *typedef* 名はすでに同じ型で宣言されています。
E0520302	[メッセージ]	Symbol has already been defined.
	[説明]	symbol はすでに定義されています。
E0520304	[メッセージ]	No instance of *名前* matches the argument list.
	[説明]	関数 " *名前* " の引数が一致しません。
E0520305	[メッセージ]	Type definition is not allowed in function return type declaration.
	[説明]	関数の戻り値の宣言において型定義はできません。
E0520306	[メッセージ]	Default argument not at end of parameter list.
	[説明]	デフォルト引数の宣言がパラメータリストの最後ではありません。
E0520307	[メッセージ]	Redefinition of default argument.
	[説明]	デフォルト引数を再定義しています。
E0520308	[メッセージ]	More than one instance of *名前* matches the argument list:
	[説明]	引数リストが一致するためオーバーロード関数 " *名前* " があいまいです。
E0520309	[メッセージ]	More than one instance of constructor *名前* matches the argument list:
	[説明]	引数リストが一致するためコンストラクタ " *名前* " があいまいです。
E0520310	[メッセージ]	Default argument of type *型1* is incompatible with parameter of type *型2*.
	[説明]	デフォルト値の " *型1* " が引数の " *型2* " に合致しません。
E0520311	[メッセージ]	Cannot overload functions distinguished by return type alone.
	[説明]	リターン型が異なる関数をオーバーロードすることはできません。

E0520312	[メッセージ]	No suitable user-defined conversion from *型1* to *型2* exists.
	[説明]	適切な利用者定義変換 " *型1*" から " *型2*" が存在しません。
E0520313	[メッセージ]	Type qualifier is not allowed on this function.
	[説明]	関数に型限定子 (const,volatile) を指定することはできません。
E0520314	[メッセージ]	Only nonstatic member functions may be virtual.
	[説明]	静的メンバ関数に virtual を指定しています。
E0520315	[メッセージ]	The object has cv-qualifiers that are not compatible with the member function.
	[説明]	オブジェクトの型限定子 (const,volatile) がメンバ関数の型限定子と合致しません。
E0520316	[メッセージ]	Program too large to compile (too many virtual functions).
	[説明]	仮想関数の数が多すぎます。
E0520317	[メッセージ]	Return type is not identical to nor covariant with return type *型* of overridden virtual function *名前* .
	[説明]	仮想関数 " *名前*" のリターン型 " *型*" が異なります。
E0520318	[メッセージ]	Override of virtual *名前* is ambiguous.
	[説明]	仮想関数 " *名前*" の置き換えがあいまいです。
E0520319	[メッセージ]	Pure specifier ("= 0") allowed only on virtual functions.
	[説明]	純粋指定子 "=0" を仮想関数以外に指定しています。
E0520320	[メッセージ]	Badly-formed pure specifier (only "= 0" is allowed).
	[説明]	純粋指定子の形式が正しくありません。"=0" だけが許されます。
E0520321	[メッセージ]	Data member initializer is not allowed.
	[説明]	データ・メンバー初期化子は許されていません。
E0520322	[メッセージ]	Object of abstract class type *型* is not allowed:
	[説明]	抽象クラス " *型*" のオブジェクトは定義できません。
E0520323	[メッセージ]	function returning abstract class *型* is not allowed:
	[説明]	抽象クラス " *型*" を返す関数は定義できません。
E0520325	[メッセージ]	inline 指定子は関数宣言のみに利用できます。
	[メッセージ]	【CC-RX】 inline specifier allowed on function declarations only.
	[説明]	【CC-RX】 inline 指定子は関数宣言のみに利用できます。
E0520326	[メッセージ]	inline is not allowed.
	[説明]	inline は許されていません。
E0520327	[メッセージ]	Invalid storage class for an inline function.
	[説明]	inline 関数の記憶クラスが不正です。
E0520328	[メッセージ]	Invalid storage class for a class member.
	[説明]	クラスメンバの記憶クラスが不正です。
E0520329	[メッセージ]	Local class member *名前* requires a definition.
	[説明]	局所クラスメンバ " *名前*" の定義がありません。

E0520330	[メッセージ]	*名前* is inaccessible.
	[説明]	*名前*をアクセスできません。
E0520332	[メッセージ]	class *型* has no copy constructor to copy a const object.
	[説明]	クラス " *型* " に const 型オブジェクトをコピーするコピーコンストラクタがありません。
E0520333	[メッセージ]	Defining an implicitly declared member function is not allowed.
	[説明]	暗黙宣言されたメンバ関数を定義することはできません。
E0520334	[メッセージ]	class *型* has no suitable copy constructor.
	[説明]	クラス " *型* " に適切なコピーコンストラクタが存在しません。
E0520335	[メッセージ]	Linkage specification is not allowed.
	[説明]	リンケージの指定は許されていません。
E0520336	[メッセージ]	Unknown external linkage specification.
	[説明]	不明な外部リンケージ指定です。
E0520337	[メッセージ]	Linkage specification is incompatible with previous "*symbol*".
	[説明]	リンケージの指定がすでにある "*symbol*" と適合しません。
E0520338	[メッセージ]	More than one instance of overloaded function *名前* has "C" linkage.
	[説明]	C リンケージを持ったオーバーロード関数 " *名前* " が複数あります。
E0520339	[メッセージ]	class *型* has more than one default constructor.
	[説明]	クラス " *型* " は複数のデフォルトコンストラクタを持っています。
E0520340	[メッセージ]	Value copied to temporary, reference to temporary used.
	[説明]	値がローカルな領域にコピーされました。ローカルな領域への参照が使用されます。
E0520341	[メッセージ]	"operator *演算子* " must be a member function.
	[説明]	演算子関数 " *演算子* " はメンバ関数でなければなりません。
E0520342	[メッセージ]	Operator may not be a static member function.
	[説明]	静的メンバ関数の演算子関数は許されません。
E0520343	[メッセージ]	No arguments allowed on user-defined conversion.
	[説明]	利用者定義変換に引数は許されません。
E0520344	[メッセージ]	Too many parameters for this operator function.
	[説明]	演算子関数の引数の数が多すぎます。
E0520345	[メッセージ]	Too few parameters for this operator function.
	[説明]	演算子関数の引数の数が足りません。
E0520346	[メッセージ]	Nonmember operator requires a parameter with class type.
	[説明]	メンバ関数でない演算子関数はクラス型を引数に持つ必要があります。
E0520347	[メッセージ]	Default argument is not allowed.
	[説明]	デフォルト引数は許されていません。

E0520348	[メッセージ]	More than one user-defined conversion from *型 1* to *型 2* applies:
	[説明]	*型 1* から " *型 2*" への利用者定義型変換があいまいです。
E0520349	[メッセージ]	No operator *演算子* matches these operands.
	[説明]	演算子関数 " *演算子* " のオペランドが一致しません。
E0520350	[メッセージ]	More than one operator *演算子* matches these operands:
	[説明]	演算子関数 " *演算子* " のオペランドがあいまいです。
E0520351	[メッセージ]	First parameter of allocation function must be of type "size_t".
	[説明]	operator new の第 1 パラメータは size_t 型でなければなりません。
E0520352	[メッセージ]	Allocation function requires "void *" return type.
	[説明]	operator new のリターン型は void * 型でなければなりません。
E0520353	[メッセージ]	Deallocation function requires "void" return type.
	[説明]	operator delete のリターン型は void 型でなければなりません。
E0520354	[メッセージ]	First parameter of deallocation function must be of type "void *".
	[説明]	operator delete の第 1 パラメータは void * 型でなければなりません。
E0520356	[メッセージ]	Type must be an object type.
	[説明]	型はオブジェクト型でなければなりません。
E0520357	[メッセージ]	Base class *xxx* has already been initialized.
	[説明]	基底クラスはすでに初期化されています。
E0520358	[メッセージ]	Base class name required -- *xxx* assumed (anachronism).
	[説明]	名前はすでに初期化されています。
E0520359	[メッセージ]	Symbol has already been initialized.
	[説明]	symbol はすでに初期化されています。
E0520360	[メッセージ]	Name of member or base class is missing.
	[説明]	メンバ名または基底クラスに誤りがあります。
E0520363	[メッセージ]	Invalid anonymous union -- nonpublic member is not allowed.
	[説明]	無名 union のメンバが公開メンバでありません。
E0520364	[メッセージ]	Invalid anonymous union -- member function is not allowed.
	[説明]	無名 union にメンバ関数は許されません。
E0520365	[メッセージ]	Anonymous union at global or namespace scope must be declared static.
	[説明]	グローバルまたは namespace スコープの無名 union は static 宣言が必要です。
E0520366	[メッセージ]	Symbol provides no initializer for:
	[説明]	symbol が初期化に使用されました :
E0520367	[メッセージ]	Implicitly generated constructor for class *型* cannot initialize:
	[説明]	暗黙に生成されたクラス " *型* " のコンストラクタを初期化することはできません。
E0520369	[メッセージ]	*名前* has an uninitialized const or reference member.
	[説明]	*名前*の const またはリファレンスメンバが初期化されていません。

E0520371	[メッセージ]	class *型* has no assignment operator to copy a const object.
	[説明]	const オブジェクトをコピーするクラス " *型* " の代入演算子関数が定義されていません。
E0520372	[メッセージ]	class *型* has no suitable assignment operator.
	[説明]	クラス " *型* " に適当な代入演算が定義されていません。
E0520373	[メッセージ]	Ambiguous assignment operator for class *型* .
	[説明]	クラス " *型* " の代入演算子関数があいまいです。
E0520375	[メッセージ]	宣言は typedef 名を必要とします。
	[メッセージ]	【CC-RX】 Declaration requires a typedef name.
	[説明]	【CC-RX】 宣言は typedef 名を必要とします。
E0520378	[メッセージ]	static is not allowed.
	[説明]	static は許されていません。
E0520380	[メッセージ]	Expression must have pointer-to-member type.
	[説明]	式はメンバへのポインタ型でなければなりません。
E0520384	[メッセージ]	No instance of overloaded *名前* matches the argument list.
	[説明]	オーバーロード関数 " *名前* " の引数リストが一致しません。
E0520386	[メッセージ]	No instance of *名前* matches the required type.
	[説明]	要求される型のオーバーロード関数 " *名前* " がありません。
E0520389	[メッセージ]	A cast to abstract class *型* is not allowed:
	[説明]	抽象クラス " *型* " へのキャストは許されません。
E0520390	[メッセージ]	Function "main" may not be called or have its address taken.
	[説明]	関数 "main" の呼び出しおよびアドレスの取得は禁止されています。
E0520391	[メッセージ]	A new-initializer may not be specified for an array.
	[説明]	配列を new によって初期化することはできません。
E0520392	[メッセージ]	Member function *名前* may not be redeclared outside its class.
	[説明]	メンバ関数 " *名前* " がクラスの外側で再宣言されました。
E0520393	[メッセージ]	Pointer to incomplete class type is not allowed.
	[説明]	不完全クラスへのポインタ型は許されません。
E0520394	[メッセージ]	Reference to local variable of enclosing function is not allowed.
	[説明]	ローカルクラスを囲む関数の局所変数へのリファレンスは許されません。
E0520397	[メッセージ]	Implicitly generated assignment operator cannot copy:
	[説明]	暗黙に生成された代入演算子関数がオブジェクトを正しくコピーすることができません。
E0520401	[メッセージ]	Destructor for base class *型* is not virtual.
	[説明]	基底クラス " *型* " のデストラクタが virtual でありません。
E0520403	[メッセージ]	Invalid redeclaration of member "symbol".
	[説明]	メンバ "symbol" の再宣言は無効です。

E0520404	[メッセージ]	Function "main" may not be declared inline.
	[説明]	関数 "main" は inline 宣言できません。
E0520405	[メッセージ]	Member function with the same name as its class must be a constructor.
	[説明]	クラス名と同じ名前のメンバ関数はコンストラクタでなければなりません。
E0520407	[メッセージ]	A destructor may not have parameters.
	[説明]	デストラクタは引数を持つことができません。
E0520408	[メッセージ]	Copy constructor for class *型1* may not have a parameter of type *型2*.
	[説明]	クラス " *型1*" のコピーコンストラクタは " *型2*" の引数を持つことはできません。
E0520409	[メッセージ]	*種別*" *シンボル名*" は不完全型 " *型名*" を返します。
	[メッセージ]	【CC-RX】 *symbol* returns incomplete type "*type*"
	[説明]	【CC-RX】 *symbol* は不完全型 "*type*" を返します。
E0520410	[メッセージ]	Protected *名前* is not accessible through a *型* pointer or object.
	[説明]	限定公開名 " *名前*" は " *型*" へのポインタやオブジェクトを経由してアクセスすることはできません。
E0520411	[メッセージ]	A parameter is not allowed.
	[説明]	引数は許されていません。
E0520412	[メッセージ]	"__asm" 宣言はここでは許されていません。
	[メッセージ]	【CC-RX】 An "__asm" declaration is not allowed here.
	[説明]	【CC-RX】 "__asm" 宣言はここでは許されていません。
E0520413	[メッセージ]	No suitable conversion function from *型1* to *型2* exists.
	[説明]	*型1* から " *型2*" への適切な変換関数が存在しません。
E0520415	[メッセージ]	No suitable constructor exists to convert from *型1* to *型2*.
	[説明]	*型1* から " *型2*" へ変換する適切なコンストラクタが存在しません。
E0520416	[メッセージ]	More than one constructor applies to convert from *型1* to *型2*:
	[説明]	*型1* から " *型2*" へ変換するコンストラクタがあいまいです。
E0520417	[メッセージ]	More than one conversion function from *型1* to *型2* applies:
	[説明]	*型1* から " *型2*" への変換関数があいまいです。
E0520418	[メッセージ]	More than one conversion function from *型* to a built-in type applies:
	[説明]	*型*から組み込み型への変換関数があいまいです。
E0520424	[メッセージ]	A constructor or destructor may not have its address taken.
	[説明]	コンストラクタ，またはデストラクタのアドレスを参照することはできません。
E0520427	[メッセージ]	Qualified name is not allowed in member declaration.
	[説明]	限定名をメンバ宣言の中で使用できません。
E0520429	[メッセージ]	The size of an array in "new" must be non-negative.
	[説明]	new で指定された配列のサイズに負の値は許されません。
E0520432	[メッセージ]	enum declaration is not allowed.
	[説明]	enum 宣言は許されていません。

E0520433	[メッセージ]	Qualifiers dropped in binding reference of type *型1* to initializer of type *型2*.
	[説明]	const/volatile 限定の型 " *型2*" が参照型 " *型1*" の初期値に指定されました。
E0520434	[メッセージ]	A reference of type *型1* (not const-qualified) cannot be initialized with a value of type *型2*.
	[説明]	const 型修飾されない型 " *型1*" へのリファレンスを " *型2*" の値で初期化できません。
E0520435	[メッセージ]	A pointer to function may not be deleted.
	[説明]	関数へのポインタを削除することはできません。
E0520436	[メッセージ]	Conversion function must be a nonstatic member function.
	[説明]	変換関数は非静的メンバ関数でなければなりません。
E0520437	[メッセージ]	Template declaration is not allowed here.
	[説明]	このスコープ内でテンプレート宣言は許されません。
E0520438	[メッセージ]	Expected a "<".
	[説明]	< が必要です。
E0520439	[メッセージ]	Expected a ">".
	[説明]	> が必要です。
E0520440	[メッセージ]	Template parameter declaration is missing.
	[説明]	テンプレートの引数宣言が正しくありません。
E0520441	[メッセージ]	Argument list for " *名前* " is missing.
	[説明]	テンプレート " *名前* " の実引数リストが正しくありません。
E0520442	[メッセージ]	Too few arguments for " *名前* ".
	[説明]	テンプレート " *名前* " の実引数が足りません。
E0520443	[メッセージ]	Too many arguments for "symbol".
	[説明]	テンプレートの実引数が多すぎます。
E0520445	[メッセージ]	*名前1* is not used in declaring the parameter types of " *名前2*".
	[説明]	テンプレート " *名前2*" の引数 " *名前1*" が使用されません。
E0520449	[メッセージ]	More than one instance of *名前* matches the required type.
	[説明]	オーバーロード関数 " *名前* " があいまいです。
E0520450	[メッセージ]	型 "long long" は標準ではありません。
	[メッセージ]	【CC-RX】 The type "long long" is nonstandard.
	[説明]	【CC-RX】 型 "long long" は標準ではありません。
E0520451	[メッセージ]	Omission of "class" is nonstandard.
	[説明]	"class" のない friend 宣言は標準形式ではありません。
E0520452	[メッセージ]	Return type may not be specified on a conversion function.
	[説明]	変換関数のリターン型が指定されていません。
E0520456	[メッセージ]	Excessive recursion at instantiation of *名前* .
	[説明]	テンプレート " *名前* " のインスタンスが再帰的に生成されます。

E0520457	[メッセージ]	*名前* is not a function or static data member.
	[説明]	*名前*が関数または静的データ・メンバではありません。
E0520458	[メッセージ]	Argument of type *型1* is incompatible with template parameter of type *型2*.
	[説明]	実引数の型 " *型1*" がテンプレートの引数 " *型2*" に合致しません。
E0520459	[メッセージ]	Initialization requiring a temporary or conversion is not allowed.
	[説明]	初期化にテンポラリや変換を要求することは許されません。
E0520460	[メッセージ]	declaration of *xxx* hides function parameter.
	[説明]	*xxx* の宣言は関数引数を隠します。
E0520461	[メッセージ]	Initial value of reference to non-const must be an lvalue.
	[説明]	const 型を持たないリファレンスの初期値は左辺値でなければなりません。
E0520463	[メッセージ]	"template" is not allowed.
	[説明]	template 指定は許されません。
E0520464	[メッセージ]	*型* is not a class template.
	[説明]	*型*がクラステンプレートではありません。
E0520466	[メッセージ]	"main" is not a valid name for a function template.
	[説明]	main は関数テンプレートの名前に使用できません。
E0520467	[メッセージ]	Invalid reference to *名前* (union/nonunion mismatch).
	[説明]	*名前*の参照が不正です。
E0520468	[メッセージ]	A template argument may not reference a local type.
	[説明]	テンプレートの実引数はローカルな型を参照できません。
E0520469	[メッセージ]	Tag kind of *xxx* is incompatible with declaration of "symbol".
	[説明]	*xxx* のタグの種類は，"symbol" の宣言と一致しません。
E0520470	[メッセージ]	The global scope has no tag named *xxx*.
	[説明]	グローバルスコープは *xxx* というタグを持っていません。
E0520471	[メッセージ]	symbol has no tag member named *xxx*.
	[説明]	symbol は *xxx* というタグメンバを持ちません。
E0520473	[メッセージ]	*名前* may be used only in pointer-to-member declaration.
	[説明]	typedef 名 " *名前* " はメンバへのポインタ型の宣言の中で使用されなければなりません。
E0520475	[メッセージ]	A template argument may not reference a non-external entity.
	[説明]	テンプレートの実引数は外部名以外を参照できません。
E0520476	[メッセージ]	Name followed by "::~" must be a class name or a type name.
	[説明]	"::~" に続く名前はクラス名または型名でなければなりません。
E0520477	[メッセージ]	Destructor name does not match name of class *型*.
	[説明]	クラス名 " *型*" とデストラクタ名が合致しません。
E0520478	[メッセージ]	Type used as destructor name does not match type *型*.
	[説明]	デストラクタ名で使われた型と " *型*" が合致しません。

E0520481	[メッセージ]	Invalid storage class for a template declaration.
	[説明]	テンプレート宣言の記憶クラス指定が正しくありません。
E0520484	[メッセージ]	Invalid explicit instantiation declaration.
	[説明]	テンプレートの実引数が不正です。
E0520485	[メッセージ]	*名前* is not an entity that can be instantiated.
	[説明]	テンプレート " *名前* " を実体化できません。
E0520486	[メッセージ]	Compiler generated *名前* cannot be explicitly instantiated.
	[説明]	コンパイラが生成した関数を実体化することはできません。
E0520487	[メッセージ]	Inline *名前* cannot be explicitly instantiated.
	[説明]	インライン関数 " *名前* " を実体化することはできません。
E0520489	[メッセージ]	*名前* cannot be instantiated -- no template definition was supplied.
	[説明]	テンプレート定義がないため " *名前* " を実体化することはできません。
E0520490	[メッセージ]	*名前* cannot be instantiated -- it has been explicitly specialized.
	[説明]	*名前*を実体化することはできません。
E0520493	[メッセージ]	No instance of *名前* matches the specified type.
	[説明]	オーバーロード関数 " *名前* " と指定された型が合致しません。
E0520494	[メッセージ]	typedef を伴う void の引数リストの宣言は標準でありません。
	[メッセージ]	【CC-RX】 Declaring a void parameter list with a typedef is nonstandard.
	[説明]	【CC-RX】 typedef を伴う void の引数リストの宣言は標準でありません。
E0520496	[メッセージ]	Template parameter *名前* may not be redeclared in this scope.
	[説明]	テンプレート引数 " *名前* " がスコープ内で再宣言されています。
E0520498	[メッセージ]	Template argument list must match the parameter list.
	[説明]	テンプレート実引数と仮引数が合致しません。
E0520500	[メッセージ]	Extra parameter of postfix "operator *xxx*" must be of type "int".
	[説明]	後置演算関数の第 2 パラメータの型は int 型でなければなりません。
E0520501	[メッセージ]	An operator name must be declared as a function.
	[説明]	演算子名は関数として宣言しなければなりません。
E0520502	[メッセージ]	Operator name is not allowed.
	[説明]	演算子名は許されません。
E0520503	[メッセージ]	*名前* cannot be specialized in the current scope.
	[説明]	スコープ内で " *名前* " があいまいです。
E0520504	[メッセージ]	Nonstandard form for taking the address of a member function.
	[説明]	メンバ関数のアドレスを取得するのは標準形式ではありません。
E0520505	[メッセージ]	Too few template parameters -- does not match previous declaration.
	[説明]	テンプレートの引数が足りません。

E0520506	[メッセージ]	Too many template parameters -- does not match previous declaration.
	[説明]	テンプレートの引数が多すぎます。
E0520507	[メッセージ]	Function template for operator delete(void *) is not allowed.
	[説明]	operator delete(void *) の関数テンプレートは許されません。
E0520508	[メッセージ]	class template and template parameter may not have the same name.
	[説明]	クラステンプレートとテンプレートの引数が同じ名前です。
E0520510	[メッセージ]	A template argument may not reference an unnamed type.
	[説明]	テンプレートの実引数が名前付けされていない型を参照しています。
E0520511	[メッセージ]	Enumerated type is not allowed.
	[説明]	列挙型は許されていません。
E0520513	[メッセージ]	型 " *型名 1*" の値は型 " *型名 2*" の実体として代入できません。
	[メッセージ]	【CC-RX】 A value of type " *型名 1*" cannot be assigned to an entity of type " *型名 2*".
	[説明]	【CC-RX】 型 " *型名 1*" の値は型 " *型名 2*" の実体として代入できません。
E0520515	[メッセージ]	Cannot convert to incomplete class *型* .
	[説明]	不完全型 " *型*" への型変換はできません。
E0520516	[メッセージ]	const object requires an initializer.
	[説明]	const のオブジェクトは初期化が必要です。
E0520517	[メッセージ]	Object has an uninitialized const or reference member.
	[説明]	オブジェクトが未初期化の const 型メンバあるいはリファレンス型メンバを持ちます。
E0520518	[メッセージ]	標準でない前処理指令です。
	[メッセージ]	【CC-RX】 Nonstandard preprocessing directive.
	[説明]	【CC-RX】 標準でない前処理指令です。
E0520519	[メッセージ]	*名前* may not have a template argument list.
	[説明]	*名前*はテンプレート実引数を持つことができません。
E0520520	[メッセージ]	集成体は "{...}" により初期化してください。
	[メッセージ]	【CC-RX】 Initialization with "{...}" expected for aggregate object.
	[説明]	【CC-RX】 集成体は "{...}" により初期化してください。
E0520521	[メッセージ]	Pointer-to-member selection class types are incompatible (*型 1* and *型 2*).
	[説明]	メンバへのポインタ型のクラスの型が " *型 1*" と " *型 2*" で合致しません。
E0520525	[メッセージ]	依存文に宣言は許されません。
	[説明]	if() の直後に "{" なしで宣言を書くことはできません。
	[メッセージ]	【CC-RX】 A dependent statement may not be a declaration.
	[説明]	【CC-RX】 if() の直後に "{" なしで宣言を書くことはできません。
E0520526	[メッセージ]	引数は void 型を持てません。
	[メッセージ]	【CC-RX】 A parameter may not have void type.
	[説明]	【CC-RX】 引数は void 型を持てません。

E0520529	[メッセージ]	This operator is not allowed in a template argument expression.
	[説明]	テンプレートの実引数式に指定された演算は許されません。
E0520530	[メッセージ]	Try block requires at least one handler/
	[説明]	try 文に対応する catch 文がありません。
E0520531	[メッセージ]	Handler requires an exception declaration.
	[説明]	catch 文の (...) には例外宣言が必要です。
E0520532	[メッセージ]	Handler is masked by default handler.
	[説明]	ハンドラはデフォルトのハンドラにマスクされました。
E0520536	[メッセージ]	Exception specification is incompatible with that of previous *名前* .
	[説明]	例外処理指定が前の指定 " *名前* " と合致しません。
E0520540	[メッセージ]	Support for exception handling is disabled.
	[説明]	例外処理を行うオプション (exception) が指定されていません。
E0520543	[メッセージ]	Non-arithmetic operation not allowed in nontype template argument.
	[説明]	対応するテンプレートの実引数に非算術型変換は許されません。
E0520544	[メッセージ]	ローカルでない変数の宣言にローカルな型が使用されました。
	[メッセージ]	【CC-RX】 Use of a local type to declare a nonlocal variable.
	[説明]	【CC-RX】 ローカルでない変数の宣言にローカルな型が使用されました。
E0520545	[メッセージ]	ローカルな型が関数宣言に使用されました。
	[メッセージ]	【CC-RX】 Use of a local type to declare a function.
	[説明]	【CC-RX】 ローカルな型が関数宣言に使用されました。
E0520546	[メッセージ]	Transfer of control bypasses initialization of:
	[説明]	初期化されないパスがあります :
E0520548	[メッセージ]	Transfer of control into an exception handler.
	[説明]	例外ハンドラ処理が実行されます。
E0520551	[メッセージ]	symbol cannot be defined in the current scope.
	[説明]	symbol はこのスコープで定義できません。
E0520555	[メッセージ]	Tag kind of *名前* is incompatible with template parameter of type *型* .
	[説明]	タグ " *名前* " の種類とテンプレートの引数の " *型* " が合致しません。
E0520556	[メッセージ]	Function template for operator new(size_t) is not allowed.
	[説明]	operator new(size_t) の関数テンプレートは許されません。
E0520558	[メッセージ]	Pointer to member of type "*type*" is not allowed.
	[説明]	型 "*type*" のメンバへのポインタは許されていません。
E0520559	[メッセージ]	Tointer to member of type *型* is not allowed.
	[説明]	メンバへのポインタ型 " *型* " が誤っています。
E0520560	[メッセージ]	symbol is reserved for future use as a keyword.
	[説明]	symbol は将来のために予約されたキーワードです。

E0520561	[メッセージ]	Invalid macro definition:
	[説明]	無効なマクロ定義です :
E0520562	[メッセージ]	Invalid macro undefinition:
	[説明]	無効なマクロ定義の取り消しです :
E0520598	[メッセージ]	A template parameter may not have void type.
	[説明]	テンプレートの引数に void 型は指定できません。
E0520599	[メッセージ]	Excessive recursive instantiation of *名前* due to instantiate-all mode.
	[説明]	instantiate-all モードの指定によってテンプレート " *名前* " のインスタンスが再帰的に生成されます。
E0520601	[メッセージ]	A throw expression may not have void type.
	[説明]	throw 式に void 型は指定できません。
E0520603	[メッセージ]	Parameter of abstract class type *型* is not allowed:
	[説明]	抽象クラス " *型* " の引数は許されません。
E0520604	[メッセージ]	Array of abstract class *型* is not allowed:
	[説明]	抽象クラス " *型* " の配列は許されません。
E0520605	[メッセージ]	Floating-point template parameter is nonstandard.
	[説明]	浮動小数点のテンプレートパラメータは標準形式ではありません。
E0520606	[メッセージ]	この pragma は宣言の直前でなければなりません。
	[メッセージ]	【CC-RX】 This pragma must immediately precede a declaration.
	[説明]	【CC-RX】 この pragma は宣言の直前でなければなりません。
E0520607	[メッセージ]	This pragma must immediately precede a statement.
	[説明]	この pragma は式の直前に記述しなければいけません。
E0520608	[メッセージ]	This pragma must immediately precede a declaration or statement.
	[説明]	この pragma は宣言または式の直前に記述しなければいけません。
E0520609	[メッセージ]	This kind of pragma may not be used here.
	[説明]	この種類の pragma はここで使用してはいけません。
E0520612	[メッセージ]	Specific definition of inline template function must precede its first use.
	[説明]	インライン指定されたテンプレート関数は呼び出しの前に定義しなければなりません。
E0520615	[メッセージ]	Parameter type involves pointer to array of unknown bound.
	[説明]	引数の型は境界が不明な配列へのポインタを含みます。
E0520616	[メッセージ]	Parameter type involves reference to array of unknown bound.
	[説明]	引数の型に要素数の指定がない配列への参照が含まれています。
E0520618	[メッセージ]	構造体か共用体に名前のないメンバがあります。
	[メッセージ]	【CC-RX】 Struct or union declares no named members.
	[説明]	【CC-RX】 構造体か共用体に名前のないメンバがあります。

E0520619	［メッセージ］	名前のないフィールドは標準ではありません。
	［メッセージ］	【CC-RX】 Nonstandard unnamed field.
	［説明］	【CC-RX】 名前のないフィールドは標準ではありません。
E0520620	［メッセージ］	Nonstandard unnamed member.
	［説明］	名前のないメンバは標準ではありません。
E0520643	［メッセージ］	restrict is not allowed.
	［説明］	restrict は許されていません。
E0520644	［メッセージ］	A pointer or reference to function type may not be qualified by "restrict".
	［説明］	関数へのポインタまたは参照型は ”restrict” によって修飾してはいけません。
E0520647	［メッセージ］	Conflicting calling convention modifiers.
	［説明］	呼び出し規則変更が競合しています。
E0520651	［メッセージ］	A calling convention may not be followed by a nested declarator.
	［説明］	呼び出し規則はネストされた宣言子に続けることはできません。
E0520654	［メッセージ］	Declaration modifiers are incompatible with previous declaration.
	［説明］	宣言修飾子が前の宣言と合致しません。
E0520656	［メッセージ］	Transfer of control into a try block.
	［説明］	外側のブロックから try ブロックに制御が移ります。
E0520658	［メッセージ］	Closing brace of template definition not found.
	［説明］	テンプレート定義の閉じ括弧がありません。
E0520660	［メッセージ］	パッキング値が不正です。
	［メッセージ］	【CC-RX】 Invalid packing alignment value.
	［説明］	【CC-RX】 パッキング値が不正です。
E0520661	［メッセージ］	整数定数がありません。
	［メッセージ］	【CC-RX】 Expected an integer constant.
	［説明］	【CC-RX】 整数定数がありません。
E0520663	［メッセージ］	Invalid source file identifier string.
	［説明］	ソース・ファイル識別文字列が不正です。
E0520664	［メッセージ］	A class template cannot be defined in a friend declaration.
	［説明］	フレンド宣言内でクラステンプレートを定義することはできません。
E0520665	［メッセージ］	asm is not allowed.
	［説明］	asm は許されていません。
E0520666	［メッセージ］	asm must be used with a function definition.
	［説明］	asm は関数定義とともに利用される必要があります。
E0520667	［メッセージ］	asm function is nonstandard.
	［説明］	asm 関数は標準ではありません。

E0520668	[メッセージ]	明示的な引数のない省略記号は標準ではありません。
	[メッセージ]	【CC-RX】 Ellipsis with no explicit parameters is nonstandard.
	[説明]	【CC-RX】 明示的な引数のない省略記号は標準ではありません。
E0520669	[メッセージ]	&... is nonstandard.
	[説明]	&... は標準ではありません。
E0520670	[メッセージ]	invalid use of "&...".
	[説明]	"&..." の使い方が不正です。
E0520673	[メッセージ]	A reference of type *型1* cannot be initialized with a value of type *型2*.
	[説明]	const/volatile 型 " *型1*" のリファレンスは " *型2*" の値で初期化できません。
E0520674	[メッセージ]	Initial value of reference to const volatile must be an lvalue.
	[説明]	const/volatile 型のリファレンスの初期値は左辺値でなければなりません。
E0520676	[メッセージ]	種別 " *シンボル名* "(宣言位置 行 *行番号*) の宣言のスコープ外で使用されました。
E0520691	[メッセージ]	*xxx*, required for copy that was eliminated, is inaccessible.
	[説明]	コピーコンストラクタにアクセスできません。
E0520692	[メッセージ]	*xxx* required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
	[説明]	コピーコンストラクタを呼び出すことができません。
E0520693	[メッセージ]	<typeinfo> must be included before typeid is used.
	[説明]	typeid を使うためには <typeinfo> をインクルードしなければなりません。
E0520694	[メッセージ]	*xxx* cannot cast away const or other type qualifiers.
	[説明]	" *名前* " のキャストで const などの属性を取り除くことはできません。
E0520695	[メッセージ]	The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *.
	[説明]	dynamic_cast の型は完全クラス型へのポインタ型またはリファレンス型か void * 型でなければなりません。
E0520696	[メッセージ]	The operand of a pointer dynamic_cast must be a pointer to a complete class type.
	[説明]	dynamic_cast ポインタのオペランドは完全クラス型へのポインタ型でなければなりません。
E0520697	[メッセージ]	The operand of a reference dynamic_cast must be an lvalue of a complete class type.
	[説明]	dynamic_cast のリファレンスのオペランドは完全クラス型の左辺値でなければなりません。
E0520698	[メッセージ]	The operand of a runtime dynamic_cast must have a polymorphic class type.
	[説明]	実行時 dynamic_cast のオペランドはポリモフィックなクラス型でなければなりません。
E0520701	[メッセージ]	An array type is not allowed here.
	[説明]	配列型はここでは許されません。
E0520702	[メッセージ]	Expected an "=".
	[説明]	"=" がありません。

E0520703	[メッセージ]	Expected a declarator in condition declaration.
	[説明]	コンディション宣言に宣言子がありません。
E0520704	[メッセージ]	*xxx*, declared in condition, may not be redeclared in this scope.
	[説明]	このスコープ内で " 名前 " を再宣言することはできません。
E0520705	[メッセージ]	Default template arguments are not allowed for function templates.
	[説明]	関数テンプレートにデフォルトの実引数を指定することはできません。
E0520706	[メッセージ]	Expected a "," or ">".
	[説明]	"," か ">" がありません。
E0520707	[メッセージ]	Expected a template parameter list.
	[説明]	テンプレートの引数リストが必要です。
E0520709	[メッセージ]	bool type is not allowed.
	[説明]	bool 型の値をデクリメントすることはできません。
E0520710	[メッセージ]	Offset of base class *名前 1* within class *名前 2* is too large.
	[説明]	クラス " *名前 2*" 内の基底クラス " *名前 1*" のサイズが大きすぎます。
E0520711	[メッセージ]	Expression must have bool type (or be convertible to bool).
	[説明]	式の型は bool 型か bool 型へ変換可能な型でなければなりません。
E0520717	[メッセージ]	The type in a const_cast must be a pointer, reference, or pointer to member to an object type.
	[説明]	const_cast の型はポインタ型，リファレンス型またはメンバへのポインタ型でなければなりません。
E0520718	[メッセージ]	A const_cast can only adjust type qualifiers; it cannot change the underlying type.
	[説明]	const_cast は const/volatile 以外の型を調整することはできません。
E0520719	[メッセージ]	mutable is not allowed.
	[説明]	mutable の指定は許されません。
E0520724	[メッセージ]	namespace definition is not allowed.
	[説明]	namespace の定義はファイルスコープまたは namespace スコープ内で許されます。
E0520725	[メッセージ]	name must be a namespace name.
	[説明]	namespace の名前が正しくありません。
E0520726	[メッセージ]	namespace alias definition is not allowed.
	[説明]	namespace の別名定義はここでは許されません。
E0520727	[メッセージ]	namespace-qualified name is required.
	[説明]	namespace の限定名が要求されます。
E0520728	[メッセージ]	A namespace name is not allowed.
	[説明]	namespace 名は許されません。
E0520730	[メッセージ]	*名前* is not a class template.
	[説明]	" *名前*" はクラステンプレートのメンバではありません。

E0520731	[メッセージ]	不完全型の配列は標準ではありません。
	[メッセージ]	【CC-RX】 Array with incomplete element type is nonstandard.
	[説明]	【CC-RX】 不完全型の配列は標準ではありません。
E0520732	[メッセージ]	Allocation operator may not be declared in a namespace.
	[説明]	operator new 関数が namespace 内で宣言されています。
E0520733	[メッセージ]	Deallocation operator may not be declared in a namespace.
	[説明]	operator delete 関数が namespace 内で宣言されています。
E0520734	[メッセージ]	*名前 1* conflicts with using-declaration of *名前 2*.
	[説明]	名前 " *名前 1*" が using 宣言名 " *名前 2*" と衝突します。
E0520735	[メッセージ]	using-declaration of *名前 1* conflicts with *名前 2*.
	[説明]	using 宣言の名前が衝突します。
E0520742	[メッセージ]	symbol has no actual member *xxx*.
	[説明]	symbol は *xxx* の実際のメンバではありません。
E0520749	[メッセージ]	A type qualifier is not allowed.
	[説明]	型指定子は許されていません。
E0520750	[メッセージ]	*名前* was used before its template was declared.
	[説明]	" *名前*" はテンプレートが宣言される前に使われました。
E0520751	[メッセージ]	Static and nonstatic member functions with same parameter types cannot be overloaded.
	[説明]	同じ引数の型を持つ静的メンバ関数と非静的メンバ関数はオーバーロードすることはできません。
E0520752	[メッセージ]	No prior declaration of "symbol".
	[説明]	symbol の宣言がありません。
E0520753	[メッセージ]	A template-id is not allowed.
	[説明]	ここではテンプレート (template 名 <template 実引数 >) は許されません。
E0520754	[メッセージ]	A class-qualified name is not allowed.
	[説明]	ここではクラス限定名は許されません。
E0520755	[メッセージ]	symbol may not be redeclared in the current scope.
	[説明]	symbol はこのスコープで再宣言できません。
E0520756	[メッセージ]	Qualified name is not allowed in namespace member declaration.
	[説明]	namespace メンバの宣言で指定された限定名は許されません。
E0520757	[メッセージ]	symbol is not a type name.
	[説明]	symbol は型名ではありません。
E0520758	[メッセージ]	Explicit instantiation is not allowed in the current scope.
	[説明]	現在のスコープ範囲でインスタンスを明示的に生成することはできません。
E0520759	[メッセージ]	*シンボル* cannot be explicitly instantiated in the current scope.
	[説明]	*シンボル*は現在のスコープで明示的にインスタンス化できません。

E0520761	[メッセージ]	typename may only be used within a template.
	[説明]	typename キーワードはテンプレート内でのみ使用できます。
E0520765	[メッセージ]	Nonstandard character at start of object-like macro definition.
	[説明]	標準でないキャラクタがオブジェクト風なマクロ定義の最初に指定されました。
E0520766	[メッセージ]	Exception specification for virtual *名前 1* is incompatible with that of overridden *名前 2*.
	[説明]	仮想関数の例外指定 " *名前 1*" が " *名前 2*" に合致しません。
E0520767	[メッセージ]	ポインタが幅の小さな整数に変換されました。
E0520768	[メッセージ]	Exception specification for implicitly declared virtual *名前 1* is incompatible with that of overridden *名前 2*.
	[説明]	コンパイラが生成する暗黙の仮想関数 " *名前 1*" の例外指定が " *名前 2*" に合致しません。
E0520769	[メッセージ]	*名前 1*, implicitly called from *名前 2*, is ambiguous.
	[説明]	operator delete の呼び出しがあいまいです。
E0520771	[メッセージ]	"explicit" is not allowed.
	[説明]	explicit はクラス宣言内のコンストラクタにのみ指定できます。
E0520772	[メッセージ]	Declaration conflicts with *xxx* (reserved class name).
	[説明]	予約されたクラス名 type_info と衝突します。
E0520773	[メッセージ]	Only "()" is allowed as initializer for array "symbol".
	[説明]	配列 "symbol" の初期化子として "()" のみが許可されています。
E0520774	[メッセージ]	"virtual" is not allowed in a function template declaration.
	[説明]	関数テンプレートに virtual 指定はできません。
E0520775	[メッセージ]	Invalid anonymous union -- class member template is not allowed.
	[説明]	無名 union の指定が正しくありません。
E0520776	[メッセージ]	Template nesting depth does not match the previous declaration of %n.
	[説明]	テンプレートのパラメータのネストが前の宣言 " 名前 " と合致しません。
E0520777	[メッセージ]	This declaration cannot have multiple "template <...>" clauses.
	[説明]	この宣言に複数のテンプレート宣言はできません。
E0520779	[メッセージ]	*xxx*, declared in for-loop initialization, may not be redeclared in this scope.
	[説明]	for 文の初期化式で宣言された " 名前 " をこのスコープ内で再宣言できません。
E0520782	[メッセージ]	Definition of virtual *名前* is required here.
	[説明]	仮想関数の定義 " *名前* " が必要です。
E0520784	[メッセージ]	A storage class is not allowed in a friend declaration.
	[説明]	フレンド宣言に記憶クラスを指定することはできません。
E0520785	[メッセージ]	Template parameter list for *名前* is not allowed in this declaration.
	[説明]	この宣言内に " *名前* " のテンプレートの引数並びは許されません。

E0520786	[メッセージ]	*名前* is not a valid member class or function template.
	[説明]	" *名前* " は有効なメンバまたは関数テンプレートではありません。
E0520787	[メッセージ]	Not a valid member class or function template declaration.
	[説明]	有効なメンバまたは関数テンプレート宣言ではありません。
E0520788	[メッセージ]	A template declaration containing a template parameter list may not be followed by an explicit specialization declaration.
	[説明]	テンプレート関数の定義の後にテンプレート引数並びを含むテンプレート宣言は指定できません。
E0520789	[メッセージ]	Explicit specialization of *名前 1* must precede the first use of *名前 2*.
	[説明]	明示的なテンプレートの実体の定義 " *名前 1*" は最初のテンプレート " *名前 2*" を使用する前になければなりません。
E0520790	[メッセージ]	Explicit specialization is not allowed in the current scope.
	[説明]	明示的なテンプレートの実体の定義はこのスコープでは許されません。
E0520791	[メッセージ]	Partial specialization of "symbol" is not allowed.
	[説明]	テンプレート " *名前* " の部分的な定義は許されません。
E0520792	[メッセージ]	*名前* is not an entity that can be explicitly specialized.
	[説明]	" *名前* " はテンプレートのインスタンスではありません。
E0520793	[メッセージ]	Explicit specialization of %n must precede its first use.
	[説明]	明示的なテンプレートの実体 " 名前 " の定義は最初の使用より前になければなりません。
E0520795	[メッセージ]	Specializing *名前* requires "template<>" syntax.
	[説明]	" *名前* " のテンプレートの実体定義は tempalte<> 形式が要求されます。
E0520799	[メッセージ]	Specializing *シンボル* without "template<>" syntax is nonstandard.
	[説明]	“template<>” なしで " *シンボル* " を特殊化するのは標準形式ではありません。
E0520800	[メッセージ]	This declaration may not have extern "C" linkage.
	[説明]	この宣言は extern "C" リンケージを持つことはできません。
E0520801	[メッセージ]	*名前* is not a class or function template name in the current scope.
	[説明]	" *名前* " はこのスコープ内ではクラステンプレートまたは関数テンプレートではありません。
E0520803	[メッセージ]	Specifying a default argument when redeclaring an already referenced function template is not allowed.
	[説明]	すでに参照された関数テンプレートを再宣言するときにデフォルト引数を指定しています。
E0520804	[メッセージ]	Cannot convert pointer to member of base class *型2* to pointer to member of derived class *型 1* -- base class is virtual.
	[説明]	仮想基底クラス " *型 1*" のメンバポインタを派生クラス " *型 2*" のメンバポインタに変換することはできません。
E0520805	[メッセージ]	Exception specification is incompatible with that of *名前* .
	[説明]	throw 例外指定は " *名前* " の例外指定と合致しません。

E0520807	[メッセージ]	Unexpected end of default argument expression.
	[説明]	デフォルト引数式が正しくありません。
E0520808	[メッセージ]	Default-initialization of reference is not allowed.
	[説明]	リファレンス型のデフォルトの初期化は許されません。
E0520809	[メッセージ]	Uninitialized "symbol" has a const member.
	[説明]	初期化されていない "symbol" が const メンバを持っています。
E0520810	[メッセージ]	Uninitialized base class *型* has a const member.
	[説明]	未初期化の基底クラス " *型* " が const 型メンバを持ちます。
E0520811	[メッセージ]	const *名前* requires an initializer -- class *型* has no explicitly declared default constructor.
	[説明]	const 型の " *名前* " には初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。
E0520812	[メッセージ]	Const object requires an initializer -- class *型* has no explicitly declared default constructor.
	[説明]	const 型オブジェクトには初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。
E0520816	[メッセージ]	関数定義において "void" の返却型を修飾することはできません。
	[メッセージ]	【CC-RX】 In a function definition a type qualifier on a "void" return type is not allowed.
	[説明]	【CC-RX】 関数定義において "void" の返却型を修飾することはできません。
E0520817	[メッセージ]	Static data member declaration is not allowed in this class.
	[説明]	局所クラスは静的データ・メンバを持つことはできません。
E0520818	[メッセージ]	Template instantiation resulted in an invalid function declaration.
	[説明]	テンプレートで実体化された関数宣言が正しくありません。
E0520819	[メッセージ]	... is not allowed.
	[説明]	... は許されていません。
E0520822	[メッセージ]	Invalid destructor name for type *型* .
	[説明]	" *型* " のデストラクタ名が正しくありません。
E0520824	[メッセージ]	Destructor reference is ambiguous -- both *名前1* and *名前2* could be used.
	[説明]	" *名前1* " と " *名前2* " が使われました。デストラクタの参照があいまいです。
E0520827	[メッセージ]	Only one member of a union may be specified in a constructor initializer list.
	[説明]	共用体の一つのメンバのみをコンストラクタの初期化で指定できます。
E0520828	[メッセージ]	Support for "new[]" and "delete[]" is disabled.
	[説明]	new[] と delete[] はサポートされていません。
E0520832	[メッセージ]	No appropriate operator delete is visible.
	[説明]	適当な operator delete 関数が見つかりません。
E0520833	[メッセージ]	Pointer or reference to incomplete type is not allowed.
	[説明]	不完全型へのポインタまたはリファレンス型は許されません。

E0520834	[メッセージ]	Invalid partial specialization -- *名前* is already fully specialized.
	[説明]	すでに特別化された " *名前* " を部分特別化しています。
E0520835	[メッセージ]	Incompatible exception specifications.
	[説明]	例外指定の型が合致しません。
E0520840	[メッセージ]	A template argument list is not allowed in a declaration of a primary template.
	[説明]	プライマリテンプレート宣言にテンプレート実引数は指定できません。
E0520841	[メッセージ]	Partial specializations may not have default template arguments.
	[説明]	部分特別化テンプレートはデフォルトのテンプレート引数を持つことはできません。
E0520842	[メッセージ]	*名前1* is not used in or cannot be deduced from the template argument list of *名前2*.
	[説明]	部分特別化テンプレート " *名前1* " は " *名前2* " のテンプレート実引数に使用されません。
E0520844	[メッセージ]	The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter.
	[説明]	部分特別化テンプレートのテンプレート実引数がテンプレート仮引数に依存する非型の実引数を含んでいます。
E0520845	[メッセージ]	This partial specialization would have been used to instantiate *名前* .
	[説明]	この部分特別化テンプレートはプライマリテンプレート " *名前* " を実体化しようとしています。
E0520846	[メッセージ]	This partial specialization would have made the instantiation of *名前* ambiguous.
	[説明]	この部分特別化テンプレートは " *名前* " の実体化があいまいになります。
E0520847	[メッセージ]	Expression must have integral or enum type.
	[説明]	式は整数型か列挙型を持つ必要があります。
E0520848	[メッセージ]	Expression must have arithmetic or enum type.
	[説明]	式は算術型か列挙型を持つ必要があります。
E0520849	[メッセージ]	Expression must have arithmetic, enum, or pointer type.
	[説明]	式は算術型か，列挙型，またはポインタ型を持つ必要があります。
E0520850	[メッセージ]	Type of cast must be integral or enum.
	[説明]	キャストの型は整数型か列挙型である必要があります。
E0520851	[メッセージ]	Type of cast must be arithmetic, enum, or pointer.
	[説明]	キャストの型は算術型か，列挙型，またはポインタ型である必要があります。
E0520852	[メッセージ]	式は完全型のオブジェクト型へのポインタである必要があります。
	[メッセージ]	【CC-RX】 Expression must be a pointer to a complete object type.
	[説明]	【CC-RX】 式は完全型のオブジェクト型へのポインタである必要があります。
E0520854	[メッセージ]	A partial specialization nontype argument must be the name of a nontype parameter or a constant.
	[説明]	一時的な特殊化の型のない引数は型のない引数の名前か低数値でなければなりません。

E0520855	[メッセージ]	Return type is not identical to return type *型* of overridden virtual function *名前*.
	[説明]	関数のリターン型がオーバーライドされた仮想関数 " *名前* " のリターン型 " *型* " と同一でありません。
E0520857	[メッセージ]	A partial specialization of a class template must be declared in the namespace of which it is a member.
	[説明]	部分特別化テンプレートはそのメンバを含む namespace の中で宣言しなければなりません。
E0520858	[メッセージ]	*名前* is a pure virtual function.
	[説明]	" *名前* " は純粋仮想関数です。
E0520859	[メッセージ]	Pure virtual *名前* has no overrider.
	[説明]	純粋仮想関数 " *名前* " はオーバーライドされません。
E0520861	[メッセージ]	入力行に不正な文字があります。
	[メッセージ]	【CC-RX】 Invalid character in input line.
	[説明]	【CC-RX】 入力行に不正な文字があります。
E0520862	[メッセージ]	関数は不完全型 " *型名* " を返します。
	[メッセージ]	【CC-RX】 Function returns incomplete type " *型名* ".
	[説明]	【CC-RX】 関数は不完全型 " *型名* " を返します。
E0520864	[メッセージ]	*名前* is not a template.
	[説明]	" *名前* " はテンプレートでありません。
E0520865	[メッセージ]	A friend declaration may not declare a partial specialization.
	[説明]	部分特別化テンプレートはフレンド宣言内で指定できません。
E0520868	[メッセージ]	Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator).
	[説明]	2 つのテンプレート実引数リストの最後に指定する ">>" は間に空白が必要です。
E0520870	[メッセージ]	不正な多バイト文字列です。
E0520871	[メッセージ]	Template instantiation resulted in unexpected function type of *型 1* (the meaning of a name may have changed since the template declaration -- the type of the template is *型 2*).
	[説明]	" *型 2* " を持つテンプレートの実体化の結果，期待されない型 " *型 1* " の関数が作られました。
E0520872	[メッセージ]	Ambiguous guiding declaration -- more than one function template *名前* matches type *型*.
	[説明]	テンプレート関数があいまいです。
E0520873	[メッセージ]	Non-integral operation not allowed in nontype template argument.
	[説明]	非型のテンプレート実引数に非整数型の演算は許されません。
E0520875	[メッセージ]	Embedded C++ does not support templates.
	[説明]	Embedded C++ 仕様はテンプレート機能をサポートしません。
E0520876	[メッセージ]	Embedded C++ does not support exception handling.
	[説明]	Embedded C++ 仕様は例外処理機能をサポートしません。

E0520877	[メッセージ]	Embedded C++ does not support namespaces.
	[説明]	Embedded C++ 仕様は namespace 機能をサポートしません。
E0520878	[メッセージ]	Embedded C++ does not support run-time type information.
	[説明]	Embedded C++ 仕様はランタイム型情報機能をサポートしません。
E0520879	[メッセージ]	Embedded C++ does not support the new cast syntax.
	[説明]	Embedded C++ 仕様は新形式のキャスト機能をサポートしません。
E0520880	[メッセージ]	Embedded C++ does not support using-declarations.
	[説明]	Embedded C++ 仕様は using 宣言機能をサポートしません。
E0520881	[メッセージ]	Embedded C++ does not support \"mutable\".
	[説明]	Embedded C++ 仕様は mutable 機能をサポートしません。
E0520882	[メッセージ]	Embedded C++ does not support multiple or virtual inheritance.
	[説明]	Embedded C++ 仕様は多重継承／仮想継承機能をサポートしません。
E0520885	[メッセージ]	*型1* cannot be used to designate constructor for *型2*.
	[説明]	" *型1*" はコンストラクタの " *型2*" で使用することはできません。
E0520886	[メッセージ]	整数定数に不正な接尾子があります。
	[説明]	整数定数に不正な接尾子（サフィックス）があります。
	[メッセージ]	【CC-RX】 Invalid suffix on integral constant.
	[説明]	【CC-RX】 整数定数に不正な接尾子（サフィックス）があります。
E0520890	[メッセージ]	Variable length array with unspecified bound is not allowed.
	[説明]	可変長の配列で整列を指定しないのは許されていません。
E0520891	[メッセージ]	An explicit template argument list is not allowed on this declaration.
	[説明]	この宣言内では明示的なテンプレート実引数は許されません。
E0520892	[メッセージ]	An entity with linkage cannot have a type involving a variable length array.
	[説明]	リンケージを伴った実体は可変長配列を持てません。
E0520893	[メッセージ]	A variable length array cannot have static storage duration.
	[説明]	可変長配列は静的な記憶域期間を持てません。
E0520894	[メッセージ]	Entity-kind " *名前*" is not a template
	[説明]	" *名前*" はテンプレートではありません。
E0520896	[メッセージ]	Expected a template argument.
	[説明]	テンプレートの実引数が期待されます。
E0520898	[メッセージ]	Nonmember operator requires a parameter with class or enum type.
	[説明]	非メンバ演算子関数にはクラスまたは列挙型の仮引数が要求されます。
E0520901	[メッセージ]	Qualifier of destructor name *型1* does not match type *型2*.
	[説明]	" *型1*" のデストラクタの限定名が " *型2*" に一致しません。
E0520915	[メッセージ]	A segment name has already been specified.
	[説明]	セグメント名はすでに指定されています。

E0520916	[メッセージ]	Cannot convert pointer to member of derived class *型1* to pointer to member of base class *型2* -- base class is virtual.
	[説明]	派生クラス " *型1*" のメンバへのポインタ型を仮想基底クラス " *型*2" のメンバへのポインタ型に変換できません。
E0520928	[メッセージ]	Incorrect use of va_start.
	[説明]	va_start の使用法が間違っています。
E0520929	[メッセージ]	Incorrect use of va_arg.
	[説明]	va_arg の使用法が間違っています。
E0520930	[メッセージ]	Incorrect use of va_end.
	[説明]	va_end の使用法が間違っています。
E0520934	[メッセージ]	A member with reference type is not allowed in a union.
	[説明]	参照型は共用体のメンバにできません。
E0520935	[メッセージ]	"typedef" はここでは指定できません。
	[メッセージ]	【CC-RX】 typedef may not be specified here.
	[説明]	【CC-RX】 "typedef" はここでは指定できません。
E0520937	[メッセージ]	A class or namespace qualified name is required.
	[説明]	クラスまたは namespace の限定名が要求されます。
E0520938	[メッセージ]	Return type "int" omitted in declaration of function "main".
	[説明]	返却型 "int" が関数 "main" の宣言で省略されました。
E0520939	[メッセージ]	Pointer-to-member representation *xxx* is too restrictive for *xxx*.
	[説明]	メンバへのポインタの宣言が正しくありません。
E0520940	[メッセージ]	void でない種別 " *シンボル名* " に return 文がありません。
E0520946	[メッセージ]	Name following "template" must be a template.
	[説明]	"template" に続く名前はメンバテンプレートでなければなりません。
E0520948	[メッセージ]	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
	[説明]	非標準形式のローカルクラスのフレンド宣言です。クラスの定義内に前方宣言がありません。
E0520951	[メッセージ]	Return type of function "main" must be "int".
	[説明]	main 関数の返却型は "int" である必要があります。
E0520952	[メッセージ]	A nontype template parameter may not have class type.
	[説明]	テンプレート仮引数にクラス型名は指定できません。
E0520953	[メッセージ]	A default template argument cannot be specified on the declaration of a member of a class template outside of its class.
	[説明]	クラステンプレートのメンバ宣言にデフォルトのテンプレート実引数を指定できません。
E0520954	[メッセージ]	A return statement is not allowed in a handler of a function try block of a constructor.
	[説明]	コンストラクタの try ブロックのハンドラ内にリターン文は許されません。

E0520955	[メッセージ]	Ordinary and extended designators cannot be combined in an initializer designation.
	[説明]	指示子が正しくありません。
E0520956	[メッセージ]	The second subscript must not be smaller than the first.
	[説明]	2 つ目の添え字は最初のものより小さくてはいけません。
E0520960	[メッセージ]	Type used as constructor name does not match type *型*.
	[説明]	コンストラクタ名として使用された型が " *型*" と一致しません。
E0520961	[メッセージ]	Use of a type with no linkage to declare a variable with linkage.
	[説明]	リンケージを持たない型がリンケージを持つ変数宣言に使用されました。
E0520962	[メッセージ]	Use of a type with no linkage to declare a function.
	[説明]	リンケージを持たない型が関数に使用されました。
E0520963	[メッセージ]	Return type may not be specified on a constructor.
	[説明]	コンストラクタにリターン型を指定できません。
E0520964	[メッセージ]	Return type may not be specified on a destructor.
	[説明]	デストラクタにリターン型を指定できません。
E0520965	[メッセージ]	Incorrectly formed universal character name.
	[説明]	間違ったユニバーサル・キャラクタ名です。
E0520966	[メッセージ]	Universal character name specifies an invalid character.
	[説明]	ユニバーサル・キャラクタ名が不正なキャラクタを指定されました。
E0520967	[メッセージ]	A universal character name cannot designate a character in the basic character set.
	[説明]	ユニバーサル・キャラクタ名は基本的なキャラクタ・セットの文字を指定できません。
E0520968	[メッセージ]	This universal character is not allowed in an identifier.
	[説明]	このユニバーサル・キャラクタは識別子として許されていません。
E0520969	[メッセージ]	The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros.
	[説明]	識別子 __VA_ARGS__ は可変個引数マクロのリストのリプレースのみ使用できます。
E0520971	[メッセージ]	Array range designators cannot be applied to dynamic initializers.
	[説明]	配列範囲名は動的初期化子に適用できません。
E0520972	[メッセージ]	Property name cannot appear here.
	[説明]	プロパティ名はここでは使用できません。
E0520975	[メッセージ]	A variable-length array type is not allowed.
	[説明]	可変長配列は許されていません。
E0520976	[メッセージ]	A compound literal is not allowed in an integral constant expression.
	[説明]	複合リテラルは整数定数式に使用できません。
E0520977	[メッセージ]	A compound literal of type "type" is not allowed.
	[説明]	複合リテラル型 "type" は許されていません。

E0520978	[メッセージ]	A template friend declaration cannot be declared in a local class.
	[説明]	テンプレートのフレンド関数は局所クラスで宣言できません。
E0520979	[メッセージ]	Ambiguous "?" operation: second operand of type *型 1* can be converted to third operand type *型 2*, and vice versa.
	[説明]	三項演算子 "?:" の第 2 式の " *型 1*" と第 3 式の " *型 2*" が互いに変換可能な型であいまいです。
E0520980	[メッセージ]	Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type.
	[説明]	オブジェクトを呼び出していますが operator() 関数または関数へのポインタ型変換関数が定義されていません。
E0520982	[メッセージ]	There is more than one way an object of type "type" can be called for the argument list:
	[説明]	実引数リストから呼ぶことができる " *型*" のオブジェクトが 2 つ以上あります。
E0520983	[メッセージ]	typedef name has already been declared (with similar type).
	[説明]	typedef 名はすでに（同じ型で）宣言されています。
E0520985	[メッセージ]	Storage class "mutable" is not allowed for anonymous unions.
	[説明]	mutable を無名共用体に指定することはできません。
E0520987	[メッセージ]	Abstract class type *型* is not allowed as catch type:
	[説明]	抽象クラスを catch で受けることはできません。
E0520988	[メッセージ]	A qualified function type cannot be used to declare a nonmember function or a static member function.
	[説明]	修飾付き関数型を非メンバ関数や static メンバ関数の宣言に使用することはできません。
E0520989	[メッセージ]	修飾された関数型は引数宣言に使用できません。
	[メッセージ]	【CC-RX】 A qualified function type cannot be used to declare a parameter.
	[説明]	【CC-RX】 修飾された関数型は引数宣言に使用できません。
E0520990	[メッセージ]	【CC-RX】 Cannot create a pointer or reference to qualified function type.
	[説明]	【CC-RX】 修飾付き関数型へのポインタ型や参照型を作ることはできません。
E0520992	[メッセージ]	不正なマクロ定義です :。
	[メッセージ]	【CC-RX】 Invalid macro definition:
	[説明]	【CC-RX】 不正なマクロ定義です。
E0520993	[メッセージ]	" *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
	[メッセージ]	【CC-RX】 Subtraction of pointer types " *型名 1*" and " *型名 2*" is nonstandard.
	[説明]	【CC-RX】 " *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
E0520994	[メッセージ]	An empty template parameter list is not allowed in a template template parameter declaration.
	[説明]	空テンプレートパラメータを持つテンプレートをテンプレートパラメータに指定することはできません。
E0520995	[メッセージ]	Expected "class".
	[説明]	テンプレートパラメータに指定するクラステンプレートはクラスを必要とします。

E0520996	[メッセージ]	The "class" keyword must be used when declaring a template template parameter.
	[説明]	テンプレートパラメータに指定するクラステンプレートは構造体ではいけません。
E0520998	[メッセージ]	A qualified name is not allowed for a friend declaration that is a function definition.
	[説明]	friend 指定付き関数定義において，名前空間の名前付き関数名を指定することはできません。
E0520999	[メッセージ]	*symbol1* is not compatible with "*symbol2*".
	[説明]	*symbol1* は "*symbol2*" と適合しません。
E0521001	[メッセージ]	Class member designated by a using-declaration must be visible in a direct base class.
	[説明]	クラスメンバの using 指定は参照可能な直接基底クラスでなければなりません。
E0521006	[メッセージ]	A template template parameter cannot have the same name as one of its template parameters.
	[説明]	テンプレートパラメータに指定するクラステンプレート名が，それ自身のテンプレートパラメータ名と同じです。
E0521007	[メッセージ]	Recursive instantiation of default argument.
	[説明]	テンプレート関数のデフォルト引数のインスタンスが再帰的に生成されます。
E0521009	[メッセージ]	*symbol* is not an entity that can be defined.
	[説明]	*symbol* は定義できる実体ではありません。
E0521010	[メッセージ]	Destructor name must be qualified.
	[説明]	不正なデストラクタ名です。
E0521011	[メッセージ]	Friend class name may not be introduced with "typename".
	[説明]	フレンドクラスの名前を “typename” に続けて記述してはいけません。
E0521012	[メッセージ]	A using-declaration may not name a constructor or destructor.
	[説明]	using 宣言でコンストラクタまたはデストラクタを指定してはいけません。
E0521013	[メッセージ]	A qualified friend template declaration must refer to a specific previously declared template.
	[説明]	限定フレンドテンプレートは参照前に定義しておく必要があります。
E0521014	[メッセージ]	Invalid specifier in class template declaration.
	[説明]	不正な指定子がクラステンプレート宣言に含まれています。
E0521015	[メッセージ]	Argument is incompatible with formal parameter.
	[説明]	実引数は仮引数と適合しません。
E0521017	[メッセージ]	Loop in sequence of "operator->" functions starting at class *xxx*.
	[説明]	operator-> が正しくありません。
E0521018	[メッセージ]	*xxx* has no member class *xxx*.
	[説明]	クラスにないメンバを使っています。
E0521019	[メッセージ]	The global scope has no class named *xxx*.
	[説明]	クラス内の名前にファイルスコープ演算子を使っています。

E0521020	[メッセージ]	Recursive instantiation of template default argument.
	[説明]	テンプレートのデフォルト引数で再帰的にインスタンスを生成します。
E0521021	[メッセージ]	Access declarations and using-declarations cannot appear in unions.
	[説明]	union で using 指定は使えません。
E0521022	[メッセージ]	*xxx* is not a class member.
	[説明]	クラスのメンバではありません。
E0521023	[メッセージ]	Nonstandard member constant declaration is not allowed.
	[説明]	標準でない const メンバ宣言は許されていません。
E0521029	[メッセージ]	Type containing an unknown-size array is not allowed.
	[説明]	サイズの不明な配列を持つ型は許されていません。
E0521030	[メッセージ]	A variable with static storage duration cannot be defined within an inline function.
	[説明]	静的変数は inline 関数で定義できません。
E0521031	[メッセージ]	An entity with internal linkage cannot be referenced within an inline function with external linkage.
	[説明]	内部リンケージを持つ識別子は外部リンケージを持つインライン関数内で参照することはできません。
E0521032	[メッセージ]	Argument type %t does not match this type-generic function macro.
	[説明]	引数の型がジェネリック関数生成マクロの型に合いません。
E0521034	[メッセージ]	Friend declaration cannot add default arguments to previous declaration.
	[説明]	フレンド関数が宣言された場合，フレンド関数の定義にデフォルト引数をいれることはできません。
E0521035	[メッセージ]	*xxx* cannot be declared in this scope.
	[説明]	このスコープではテンプレートを宣言することができません。
E0521036	[メッセージ]	予約語 " *シンボル* " は関数の中でのみ使用できます。
	[メッセージ]	【CC-RX】 The reserved identifier *シンボル* may only be used inside a function.
	[説明]	【CC-RX】 予約語 " *シンボル* " は関数の中でのみ使用できます。
E0521037	[メッセージ]	This universal character cannot begin an identifierl.
	[説明]	このユニバーサル・キャラクタは識別子の先頭に使用できません。
E0521038	[メッセージ]	Expected a string literal.
	[説明]	文字列リテラルがありません。
E0521039	[メッセージ]	Unrecognized STDC pragma.
	[説明]	認識されない STDC pragma です。
E0521040	[メッセージ]	Expected "ON", "OFF", or "DEFAULT".
	[説明]	"ON", "OFF" または "DEFAULT" がありません。
E0521041	[メッセージ]	A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.
	[説明]	STDC pragma はグローバルスコープ内の宣言の間か，何かの文またはブロックスコープの宣言の前にのみ置くことがができます。

E0521042	[メッセージ]	Incorrect use of va_copy.
	[説明]	va_copy の使用法が間違っています。
E0521043	[メッセージ]	*xxx* can only be used with floating-point types.
	[説明]	*xxx* は浮動小数点型にのみ使用できます。
E0521044	[メッセージ]	Complex type is not allowed.
	[説明]	複素数型は許されていません。
E0521045	[メッセージ]	Invalid designator kind.
	[説明]	指示子の種類が不正です。
E0521047	[メッセージ]	Complex floating-point operation result is out of range.
	[説明]	複素数型浮動小数点演算の結果が表現可能な値の範囲を越えました。
E0521048	[メッセージ]	Conversion between real and imaginary yields zero.
	[説明]	実数と虚数の相互変換後の値が 0 になりました。
E0521049	[メッセージ]	An initializer cannot be specified for a flexible array member.
	[説明]	可変長配列メンバに初期化子を指定することはできません。
E0521051	[メッセージ]	Standard requires that "symbol" be given a type by a subsequent declaration ("int" assumed).
	[説明]	symbol の型は直後の宣言により与えられなければなりません（"int" に仮定されます）。
E0521052	[メッセージ]	A definition is required for inline "symbol".
	[説明]	インライン関数 "symbol" には定義が必要です。
E0521054	[メッセージ]	A floating-point type must be included in the type specifier for a _Complex or _Imaginary type.
	[説明]	浮動小数点型は複素数または虚数型の指定子に含まれてなければいけません。
E0521055	[メッセージ]	Types cannot be declared in anonymous unions.
	[説明]	型を無名共用体内で宣言することはできません。
E0521056	[メッセージ]	Returning pointer to local variable.
	[説明]	ローカル変数へのポインタを返しています。
E0521057	[メッセージ]	Returning pointer to local temporary.
	[説明]	ローカルな領域へのポインタを返しています。
E0521061	[メッセージ]	Declaration of "symbol" is incompatible with a declaration in another translation unit.
	[説明]	“ シンボル名 ” の宣言はもう一つの翻訳単位内の宣言と互換性がありません。
E0521062	[メッセージ]	The other declaration is %p.
	[説明]	別の宣言があります。
E0521065	[メッセージ]	A field declaration cannot have a type involving a variable length array.
	[説明]	フィールドの宣言は可変長の配列を含む型を持てません。

E0521066	［メッセージ］	" *シンボル名* " の宣言は *ファイル名* のコンパイル中は異なる意味を持ちます。
	［メッセージ］	【CC-RX】 Declaration of " *シンボル名* " had a different meaning during compilation of *ファイル名* .
	［説明］	【CC-RX】 " *シンボル名* " の宣言は *ファイル名* のコンパイル中は異なる意味を持ちます。
E0521067	［メッセージ］	Eexpected "template".
	［説明］	"template" がありません。
E0521072	［メッセージ］	宣言はラベルを持てません。
	［メッセージ］	【CC-RX】 A declaration cannot have a label.
	［説明］	【CC-RX】 宣言はラベルを持てません。
E0521075	［メッセージ］	" *シンボル名* " は *xxx* のコンパイル時にすでに定義されています。
	［メッセージ］	【CC-RX】 *シンボル名* already defined during compilation of *xxx*.
	［説明］	【CC-RX】 " *シンボル名* " は *xxx* のコンパイル時にすでに定義されています。
E0521076	［メッセージ］	" *シンボル名* " は他のコンパイル単位ですでに定義されています。
	［メッセージ］	【CC-RX】 *シンボル名* already defined in another translation unit.
	［説明］	【CC-RX】 " *シンボル名* " はほかのコンパイル単位ですでに定義されています。
E0521081	［メッセージ］	A field with the same name as its class cannot be declared in a class with a user-declared constructor.
	［説明］	クラス名と同じ名前のメンバを宣言することはできません。
E0521086	［メッセージ］	The object has cv-qualifiers that are not compatible with the member "*symbol*".
	［説明］	オブジェクトはメンバ "*symbol*" と合致しない const または volatile 修飾子を持ちます。
E0521087	［メッセージ］	No instance of *xxx* matches the argument list and object (the object has cv-qualifiers that prevent a match).
	［説明］	" *クラス名* " のインスタンスは引数リストとオブジェクトと合致しません（オブジェクトの持つ CV 修飾子が合致を抑止しています）。
E0521088	［メッセージ］	An attribute specifies a mode incompatible with *xxx*.
	［説明］	幅が指定された型がありません。
E0521089	［メッセージ］	There is no type with the width specified.
	［説明］	指定された幅を持つ型がありません。
E0521139	［メッセージ］	The "template" keyword used for syntactic disambiguation may only be used within a template.
	［説明］	キーワード "template" を構文上のあいまいさを解消するのに使用できるのは template 内のみです。
E0521144	［メッセージ］	Storage class must be auto or register.
	［説明］	記憶クラスは auto または register でなければいけません。
E0521146	［メッセージ］	*xxx* is not a base class member.
	［説明］	基底クラスのメンバではありません。

E0521158	［メッセージ］	返却型 void は修飾できません。
	［メッセージ］	【CC-RX】 void return type cannot be qualified.
	［説明］	【CC-RX】 void の返却型は修飾できません。
E0521161	［メッセージ］	A member template corresponding to *xxx* is declared as a template of a different kind in another translation unit.
	［説明］	テンプレート宣言が他コンパイル単位と異なっています。
E0521163	［メッセージ］	va_start should only appear in a function with an ellipsis parameter.
	［説明］	va_start が使用されるのは省略記号を引数とする関数のみです。
E0521201	［メッセージ］	typedef *xxx* may not be used in an elaborated type specifier.
	［説明］	typedef *xxx* は記述型指定子では使用できません。
E0521203	［メッセージ］	Parameter *引数名* may not be redeclared in a catch clause of function try block.
	［説明］	" *引数名* " を try ブロックの catch 句の中で再宣言してはいけません。
E0521204	［メッセージ］	The initial explicit specialization of *xxx* must be declared in the namespace containing the template.
	［説明］	シンボルに対する最初の明示的な特殊化はテンプレートを含む名前空間の中に宣言されなければいけません。
E0521206	［メッセージ］	"template" must be followed by an identifier.
	［説明］	"template" の後には識別子が必要です。
E0521212	［メッセージ］	This pragma cannot be used in a _Pragma operator (a #pragma directive must be used).
	［説明］	このプラグマは _Pragma operator 内では使用できません（#pragma ディレクティブを使用してください）。
E0521227	［メッセージ］	Transfer of control into a statement expression is not allowed.
	［説明］	文式中への制御の移動は許されていません。
E0521229	［メッセージ］	This statement is not allowed inside of a statement expression.
	［説明］	この文は文式の中では許されていません。
E0521230	［メッセージ］	Anon-POD class definition is not allowed inside of a statement expression.
	［説明］	非 POD クラスは式文内に定義できません。
E0521254	［メッセージ］	" *型* " のサイズまたは複雑さのため内部計算時にオーバフローを起こしました。
	［メッセージ］	【CC-RX】 Integer overflow in internal computation due to size or complexity of " *型* ".
	［説明］	【CC-RX】 " *型* " のサイズまたは複雑さのため内部計算時にオーバフローを起こしました。
E0521255	［メッセージ］	整数が内部演算時にオーバフローを起こしました。
E0521273	［メッセージ］	Alignment-of operator applied to incomplete type.
	［説明］	オペレータの整列条件が不完全な型に対して適用されました。
E0521280	［メッセージ］	Conversion from inaccessible base class *xxx* is not allowed.s
	［説明］	派生クラスにプライベートで継承された基底クラス型のポインタを継承クラス型のポインタへ変換することはできません。

E0521282	[メッセージ]	種類の異なる文字を持つ文字列リテラルは連結できません。
	[メッセージ]	【CC-RX】 種類の異なる文字を持つ文字列リテラルは連結できません。
	[説明]	【CC-RX】 String literals with different character kinds cannot be concatenated.
E0521291	[メッセージ]	A non-POD class type cannot be fetched by va_arg.
	[説明]	非 POD 型のクラスは va_arg によって取得することができません。
E0521292	[メッセージ]	The 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal.
	[説明]	型接尾語 'u' または 'U' は固定小数点リテラル中の型接尾語 'l' または 'L' の前にある必要があります。
E0521295	[メッセージ]	Fixed-point constant is out of range.
	[説明]	固定小数点定数が範囲を越えています。
E0521303	[メッセージ]	Expression must have integral, enum, or fixed-point type.
	[説明]	式は整数か列挙対または固定小数点型である必要があります。
E0521304	[メッセージ]	Expression must have integral or fixed-point type.
	[説明]	式は整数か固定小数点型である必要があります。
E0521311	[メッセージ]	Fixed-point types have no classification.
	[説明]	浮動小数点型の区分がありません。
E0521312	[メッセージ]	A template parameter may not have fixed-point type.
	[説明]	テンプレート引数には固定小数点型を指定できません。
E0521313	[メッセージ]	Hexadecimal floating-point constants are not allowed.
	[説明]	16 進の浮動小数点定数は許されていません。
E0521315	[メッセージ]	Floating-point value does not fit in required fixed-point type.
	[説明]	浮動小数値が要求された浮動小数点型に適していません。
E0521317	[メッセージ]	Fixed-point conversion resulted in a change of sign.
	[説明]	負の整数値を固定小数点型へ変換した結果，正の値になりました。
E0521318	[メッセージ]	Integer value does not fit in required fixed-point type.
	[説明]	整数値が要求された固定小数点型に適していません。
E0521319	[メッセージ]	Fixed-point operation result is out of range.
	[説明]	固定小数点の処理が範囲を越えました。
E0521320	[メッセージ]	Multiple named address spaces.
	[説明]	同一の名前アドレス空間が複数存在します。
E0521321	[メッセージ]	Variable with automatic storage duration cannot be stored in a named address space.
	[説明]	局所的なスコープを持つ変数は名前付きアドレス空間に保持することはできません。
E0521322	[メッセージ]	Type cannot be qualified with named address space.
	[説明]	名前付きアドレス空間によって型を識別することはできません。

E0521323	[メッセージ]	Function type cannot be qualified with named address space.
	[説明]	名前付きアドレス空間によって関数型を識別することはできません。
E0521324	[メッセージ]	Field type cannot be qualified with named address space.
	[説明]	フィールド型は名前付き空間によって識別することはできません。
E0521325	[メッセージ]	Fixed-point value does not fit in required floating-point type.
	[説明]	固定小数点値は要求された浮動小数点型に収まりません。
E0521326	[メッセージ]	Fixed-point value does not fit in required integer type.
	[説明]	固定小数点値は要求された整数型に収まりません。
E0521327	[メッセージ]	Value does not fit in required fixed-point type.
	[説明]	値は要求された固定小数点値に収まりません。
E0521344	[メッセージ]	A named address space qualifier is not allowed here.
	[説明]	名前付きアドレス空間識別子はここで使用できません。
E0521345	[メッセージ]	An empty initializer is invalid for an array with unspecified bound.
	[説明]	バウンドが指定されていない配列の初期化への空の初期化子は不正です。
E0521348	[メッセージ]	Declaration hides "*symbol*".
	[説明]	宣言は "*symbol*" を隠します。
E0521349	[メッセージ]	A parameter cannot be allocated in a named address space.
	[説明]	引数は名前付きアドレス空間に配置できません。
E0521350	[メッセージ]	Invalid suffix on fixed-point or floating-point constant.
	[説明]	不正な接尾辞が固定または浮動小数点定数についています。
E0521351	[メッセージ]	A register variable cannot be allocated in a named address space.
	[説明]	レジスタ変数は名前付きアドレス空間に配置できません。
E0521352	[メッセージ]	Expected "SAT" or "DEFAULT".
	[説明]	"SAT" か "DEFAULT" がありません。
E0521355	[メッセージ]	A function return type cannot be qualified with a named address space.
	[説明]	関数の戻り値を名前付きアドレス空間で修飾することはできません。
E0521365	[メッセージ]	Named-register variables cannot have void type.
	[説明]	名前付きレジスタ変数は void 型にできません。
E0521372	[メッセージ]	Nonstandard qualified name in global scope declaration.
	[説明]	非標準形式で修飾した名前がグローバルなスコープに宣言されています。
E0521380	[メッセージ]	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
	[説明]	仮想関数の定義がありません。また，無名空間のメンバであるため，それ以外の場所で定義することができません。

E0521381	[メッセージ]	キャリッジ・リターン文字がコメントおよび文字定数 / 文字列リテラルの外にありました。
	[説明]	キャリッジ・リターン文字 ('¥r') がコメントおよび文字定数／文字列リテラルの外にありました。
	[メッセージ]	【CC-RX】 Carriage return character in source line outside of comment or character/ string literal.
	[説明]	【CC-RX】 改行文字がコメントまたは文字列リテラル以外のところにあります。
E0521382	[メッセージ]	Expression must have fixed-point type.
	[説明]	式は固定小数点型である必要があります。
E0521398	[メッセージ]	Invalid member for anonymous member class -- class *xxx* has a disallowed member function.
	[説明]	無名のメンバクラスに対して不正なメンバ関数を宣言しています。
E0521403	[メッセージ]	A variable-length array is not allowed in a function return type.
	[説明]	可変長の配列は関数の返却値にできません。
E0521404	[メッセージ]	Variable-length array type is not allowed in pointer to member of type "*type*".
	[説明]	可変長配列型は "*type*" 型のメンバへのポインタではできません。
E0521405	[メッセージ]	The result of a statement expression cannot have a type involving a variable-length array.
	[説明]	文式の結果は可変長配列に関する型を持てません。
E0521420	[メッセージ]	いくつかの列挙子はその列挙型の潜在的な整数型で表現できません。
	[メッセージ]	【CC-RX】 Some enumerator values cannot be represented by the integral type underlying the enum type.
	[説明]	【CC-RX】 いくつかの列挙子はその列挙型の潜在的な整数型で表現できません。
E0521424	[メッセージ]	Second operand of offsetof must be a field.
	[説明]	2 つ目の offsetof のオペランドはフィールドである必要があります。
E0521425	[メッセージ]	Second operand of offsetof may not be a bit field.
	[説明]	2 つ目の offsetof のオペランドはビットフィールドではなりません。
E0521436	[メッセージ]	*xxx* is only allowed in C.
	[説明]	*xxx* は C 言語のみで許されています。
E0521437	[メッセージ]	__ptr32 and __ptr64 must follow a "*".
	[説明]	__ptr32 と __ptr64 は "*" に続く必要があります。
E0521441	[メッセージ]	Complex integral types are not supported.
	[説明]	複素数型はサポートされていません。
E0521442	[メッセージ]	__real and __imag can only be applied to complex values.
	[説明]	__real と __imag は複素数値にのみ使用できます。
E0521445	[メッセージ]	Invalid redefinition of "*symbol*".
	[説明]	*symbol* の不正な再定義です。
E0521534	[メッセージ]	Duplicate function modifier.
	[説明]	関数修飾子が重複しています。

E0521535	[メッセージ]	Invalid character for char16_t literal.
	[説明]	char16_t リテラルが不正です。
E0521536	[メッセージ]	__LPREFIX cannot be applied to char16_t or char32_t literals.
	[説明]	__LPREFIX は char16_t か char32_t のリテラルに指定できません。
E0521537	[メッセージ]	Unrecognized calling convention *xxx* must be one of:
	[説明]	許されていないコーリングコンベンション *xxx* です。以下の一つである必要があります :
E0521539	[メッセージ]	Option "--uliterals" can be used only when compiling C.
	[説明]	--uliterals オプションは C のコンパイル時にのみ使用できます。
E0521542	[メッセージ]	Some enumerator constants cannot be represented by "type".
	[説明]	いくつかの列挙子定数は "type" で表現できません。
E0521543	[メッセージ]	*xxx* not allowed in current mode.
	[説明]	*xxx* は現在のモードで許されていません。
E0521557	[メッセージ]	Alias creates cycle of aliased entities.
	[説明]	エイリアスがエイリアスされた実体に循環しています。
E0521558	[メッセージ]	Subscript must be constant.
	[説明]	添え字は定数である必要があります。
E0521574	[メッセージ]	Static assertion failed with *xxx*.
	[説明]	静的なアサーションは失敗しました。
E0521576	[メッセージ]	Field name resolves to more than one offset -- see "*symbol1*" and "*symbol2*".
	[説明]	フィールド名は複数のオフセットがあります -- "*symbol1*" と "*symbol2*" を見てください。
E0521577	[メッセージ]	*xxx* is not a field name.
	[説明]	*xxx* はフィールド名ではありません。
E0521578	[メッセージ]	case ラベル値はすでにこの switch の行*行番号*で現れています。
	[メッセージ]	【CC-RX】 case label value has already appeared in this switch *行番号*
	[説明]	【CC-RX】 case ラベル値はすでにこの switch の行*行番号*で現れています。
E0521582	[メッセージ]	The option to list macro definitions may not be specified when compiling more than one translation unit.
	[説明]	マクロ定義を記載するオプションは複数ファイルコンパイル時に指定できません。
E0521583	[メッセージ]	Unexpected parenthesis after declaration of "symbol" (malformed parameter list or invalid initializer?).
	[説明]	symbol の宣言の後に予期しないかっこがあります（正常でない引数リストか不正な初期化子の可能性があります）。
E0521584	[メッセージ]	文字列リテラルをかっこで囲むことは標準ではありません。
	[メッセージ]	【CC-RX】 Parentheses around a string initializer are nonstandard.
	[説明]	【CC-RX】 文字列リテラルをかっこで囲むことは標準ではありません。

E0521586	[メッセージ]	A variable declared with an auto type specifier cannot appear in its own initializer.
	[説明]	auto 型を指定され宣言された変数はそれ自身の初期化子には指定できません。
E0521587	[メッセージ]	Cannot deduce "auto" type.
	[説明]	auto 型を推測できません。
E0521588	[メッセージ]	Initialization with "{...}" is not allowed for "auto" type.
	[説明]	{...} を伴った初期化は "auto" 型に指定できません。
E0521589	[メッセージ]	auto type cannot appear in top-level array type.
	[説明]	auto 型はトップ・レベルの配列型に指定できません。
E0521590	[メッセージ]	auto type cannot appear in top-level function type.
	[説明]	auto 型はトップ・レベルの関数型には指定できません。
E0521593	[メッセージ]	Cannot deduce "auto" type (initializer required).
	[説明]	auto 型を推測できません（初期化子が必要です）。
E0521596	[メッセージ]	Invalid use of a type qualifier.
	[説明]	不正な型修飾子の使用です。
E0521597	[メッセージ]	A union cannot be abstract or sealed.
	[説明]	共用体に abstract または sealed は指定できません。
E0521598	[メッセージ]	auto is not allowed here.
	[説明]	auto はここでは許されていません。
E0521602	[メッセージ]	不完全型をメンバにもつ構造体 / 共用体変数 " *変数名* " はセクションを指定できません 。
E0521603	[メッセージ]	不完全型の変数 " *変数名* " はセクションを指定できません。
E0521604	[メッセージ]	#pragma section で指定された再配置属性が不正です。
E0521605	[メッセージ]	#pragma *文字列* の文法が不正です。
E0521606	[メッセージ]	関数 " *関数名* " はすでに別のセクションが指定されています。
	[説明]	関数 " *関数名* " はすでに別のセクションが指定されています。異なるセクションを指定することはできません。
E0521608	[メッセージ]	#pragma asm は関数の外に記述できません。
E0521609	[メッセージ]	#pragma asm が #pragma endasm によって閉じられていません。
E0521610	[メッセージ]	#pragma asm なしに #pragma endasm が記述されています。
E0521612	[メッセージ]	割り込み要求名 " *要求名* " のハンドラはすでに定義されています。
E0521613	[メッセージ]	このデバイスは割り込み要求名 " *要求名* " をサポートしていません。
E0521614	[メッセージ]	同じ関数で別の配置方法またはオプション指定の #pragma interrupt があります。
E0521615	[メッセージ]	関数 " *関数名* " は別の #pragma smart_correct で指定されています。
	[説明]	関数 " *関数名* " はすでに別の #pragma smart_correct で指定されています。
E0521616	[メッセージ]	*種別* " *シンボル名* " はすでに別のセクション指定で extern 宣言されています。
E0521617	[メッセージ]	*種別* " *シンボル名* " はすでに別のセクションが指定されています。

E0521618	[メッセージ]	*種別* " *シンボル名* " はすでにセクションが指定されています。新たにセクションなしで宣言することはできません。
E0521619	[メッセージ]	*種別* " *シンボル名* " はすでにセクションなしで宣言されています。新たにセクション指定することはできません。
E0521620	[メッセージ]	組み込み関数 " *関数名* ()" の引数の値が範囲を越えています。" *最小値～最大値* " が指定できます。
E0521621	[メッセージ]	I/O レジスタ " *レジスタ名* " は書き込みができません。
E0521622	[メッセージ]	I/O レジスタ " *レジスタ名* " は読み出しができません。
E0521623	[メッセージ]	デバイスの指定がない場合*拡張機能指定*の機能は使用できません。
E0521624	[メッセージ]	__set_il() の第二引数は文字列定数である必要があります。
E0521625	[メッセージ]	割り込み要求名 " *要求名* " の割り込みレベルは設定できません。
E0521626	[メッセージ]	*指定文字列*が関数 " *関数名* " に指定されました。以前の #pragma inline 指定を無視します。
E0521627	[メッセージ]	#pragma smart_correct に指定された関数が同じです。
E0521628	[メッセージ]	#pragma smart_correct に指定された関数 " *関数名* " が定義されていません。
E0521629	[メッセージ]	シンボル・ファイル " *ファイル名* " をオープンすることができません。
E0521630	[メッセージ]	シンボル・ファイル " *ファイル名* " をクローズすることができません。
E0521631	[メッセージ]	シンボル・ファイル中に構文エラーがありました。
E0521632	[メッセージ]	" *文字列* " は認識できない情報行です。無視します。
E0521633	[メッセージ]	セクション名が指定されていません。
E0521634	[メッセージ]	不正なセクション名 " *セクション名* " が指定されています。
E0521635	[メッセージ]	変数 " *変数名* " にはシンボル・ファイル中ですでにセクション " *セクション名* " が指定されています。後の指定を無視します。
E0521636	[メッセージ]	変数 " *変数名* " にはシンボル・ファイル中ですでにセクション " *セクション名* " が指定されています。#pragma による指定を無視します。
E0521637	[メッセージ]	2 進定数が不正です。
E0521638	[メッセージ]	*特殊関数名* () の第一引数は数値定数である必要があります。
E0521639	[メッセージ]	関数 " *関数名* " は direct 指定で割り込み関数に指定されています。セクション指定することはできません。
E0521640	[メッセージ]	関数 " *関数名* " は #pragma text によりセクション指定されています。direct 指定で割り込み関数にすることはできません。
E0521641	[メッセージ]	FE レベル割り込みは現在サポートされていません。
E0521642	[メッセージ]	再配置属性 " *属性* " のセクションは、セクション名指定ができません。
E0521643	[メッセージ]	"direct" 指定された割り込み関数は複数の割り込み要求名に指定できません。
E0521644	[メッセージ]	デバイスの例外ハンドラ縮小機能が有効になっています。EI レベル・マスカブル割り込みのハンドラアドレスが重複する可能性があります。
E0521645	[メッセージ]	void 型ではない関数 " *関数名* " は割り込み関数にできません。
E0521646	[メッセージ]	割り込み要求名に NO_VECT を指定した場合、"direct" 指定できません。
E0521647	[メッセージ]	*文字列*はここでは許されていません。
E0521648	[メッセージ]	*種別*関数 " *関数名* " は呼び出すことができません。

E0521649	［メッセージ］	*文字列 1* は、*文字列 2* と同時に使用できません。
	［説明］	*文字列 1* の機能と，*文字列 2* の機能は同時に使用することができません。
	［メッセージ］	【CC-RX】 White space is required between the macro name *xxx* and its replacement text.
	［対処方法］	【CC-RX】 マクロ名とその置換テキストの間に空白を入れて区切ってください。
E0521650	［メッセージ］	*種別* " *シンボル名* " は、すでに別な #pragma pic/nopic 指定で宣言されています。
	［説明］	*種別* " *シンボル名* " に対して矛盾する #pragma pin/nopic 指定がされています。
E0523005	［メッセージ］	#pragma の構文が不正です。
	［説明］	#pragma の構文はフォーマットに合わせて書いてください。
	［メッセージ］	【CC-RX】 Invalid pragma declaration.
	［説明］	【CC-RX】 #pragma の構文はフォーマットに合わせて書いてください。
E0523006	［メッセージ］	このシンボルは既に他の #pragma 指定がされています。
	［説明］	1 つのシンボルに対して，同時指定不可能な #pragma を 2 個以上指定しています。
	［メッセージ］	【CC-RX】 "*symbol name*" has already been specified by other pragma.
	［説明］	【CC-RX】 1 つのシンボルに対して，同時指定不可能な #pragma を 2 個以上指定しています。
E0523007	［メッセージ］	シンボル定義後の宣言にのみ #pragma 指定することはできません。
	［説明］	#pragma は対象のシンボル定義よりも先に宣言してください。
	［メッセージ］	【CC-RX】 Pragma may not be specified after definition
	［説明］	【CC-RX】 #pragma は対象のシンボル定義よりも先に宣言してください。
E0523008	［メッセージ］	不正な #pragma を指定しました。
	［説明］	このシンボルに対して，この #pragma を指定することはできません。
	［メッセージ］	【CC-RX】 Invalid kind of pragma is specified to this symbol
	［説明］	【CC-RX】 このシンボルに対して，この #pragma を指定することはできません。
E0523018	［メッセージ］	メンバに対して near または far が指定されています。
	［説明］	構造体，および共用体の定義時に，メンバに対して __near，および __far を指定することはできません。
E0523042	［メッセージ］	SuperH コンパイラとの互換性に影響ある " *機能項目* "(オプションや #pragma など) が使用されています。
	［説明］	SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
	［メッセージ］	【CC-RX】 Using "*function item*" function at influence the code generation of "SuperH" compiler
	［説明］	【CC-RX】 SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
E0523057	［メッセージ］	セクションの属性指定文字が不正です。
	［説明］	セクションの属性指定文字として，使用不可能な文字列を使用しました。
	［メッセージ］	【CC-RX】 Illegal section specified.
	［説明］	【CC-RX】 セクションの属性指定文字として，使用不可能な文字列を使用しました。

E0523058	［メッセージ］	#pragma section の構文が正しくありません。
	［説明］	#pragma section の構文が文法に違反しています。
	［メッセージ］	【CC-RX】 Illegal #pragma section syntax
	［説明］	【CC-RX】 #pragma section の構文が文法に違反しています。
E0523059	［メッセージ］	#pragma section text の後にはセクション名が必要です。
	［説明］	#pragma section の構文が正しくありません。
	［メッセージ］	【CC-RX】 Cannot change text section.
	［説明］	【CC-RX】 #pragma section の構文が正しくありません。
E0523061	［メッセージ］	Argument is incompatible with formal parameter of intrinsic function.
	［説明］	実引数は組み込み関数の仮引数と適合しません。
E0523062	［メッセージ］	Return value type does not match the intrinsic function type.
	［説明］	返却値の型が組み込み関数の型と合っていません。
E0523065	［メッセージ］	【CC-RX】 " *文字列* " has no effect in this version
	［説明］	【CC-RX】 このバージョンでは " *文字列* " は無効です。
	［メッセージ］	【CC-RH】 ビット・フィールドの初期値にアドレス定数を記述できません。
	［対処方法］	【CC-RH】 ビット・フィールドの初期値にアドレス定数を書かないでください。
E0523066	［メッセージ］	このセクションは現在のオプションでは使用できません。
	［メッセージ］	【CC-RX】 The combination of the option and section specification is inaccurate.
	［説明］	【CC-RX】 このセクションは現在のオプションでは使用できません。
E0523067	［メッセージ］	宣言子のネストが深すぎます。
	［説明］	宣言子のネストが深すぎます。
	［対処方法］	処理系限界以上のネストを書かないでください。
E0523069	［メッセージ］	pm 番号を複数使うことはできません。
E0523070	［メッセージ］	"cmn" 指定変数は、r0 相対のみでアクセスできます。
E0523071	［メッセージ］	"cmn" 指定関数がアクセス可能な静的変数は r0 相対に限ります。
E0523072	［メッセージ］	"cmn" 指定関数が呼出し可能な "pmodule" 指定関数は、"cmn" 指定に限ります。
E0523074	［メッセージ］	" *関数名* " は #pragma rtos_interrupt と共に使用できません。
	［説明］	" *関数名* " に #pragma rtos_interrupt を指定することはできません。
E0523075	［メッセージ］	near/far 属性とアドレスの組み合わせが不適切です。
	［説明］	#pragma address で指定したアドレスが，変数に指定された __near，および __far の属性と矛盾した配置になっています。
E0532002	［メッセージ］	浮動小数点数の演算時に例外 *例外名* が発生しました。
E0541004	［メッセージ］	__sectop/__secend で取得した値と定数との加減算をしています。
E0541240	［メッセージ］	" *名前* " セクションの命名に誤りがあります。用途の異なるセクションに同じ名前を付けています。
E0541854	［メッセージ］	#pragma address で指定したアドレスが不正です。
	［説明］	異なる変数に対して，同一アドレスを指定しています。

E0544003	［メッセージ］	The size of " *セクション名* " section exceeds the limit.
	［説明］	" *セクション名* " セクションのサイズが上限値を越えています。
E0544240	［メッセージ］	Illegal naming of section " *セクション名* ".
	［説明］	セクションの命名に誤りがあります。用途の異なるセクションに同じ名前が付いています。
E0544854	［メッセージ］	Illegal address was specified with #pragma address.
	［説明］	#pragma address によるアドレスの指定が以下のいずれかに該当しています。 (1) 異なる変数に対して，同一アドレスを指定している。 (2) 異なる変数に対して，変数のアドレスが重なっている。
E0550200	［メッセージ］	整列条件の指定に誤りがあります。
	［対処方法］	整列条件の指定を確認してください。
E0550201	［メッセージ］	扱うことのできない文字が現れました。
	［対処方法］	文字を確認してください。
E0550202	［メッセージ］	式の構成に誤りがあります。
	［対処方法］	式を確認してください。
E0550203	［メッセージ］	式の要素 *string* に誤りがあります。
	［対処方法］	式の要素を確認してください。
E0550204	［メッセージ］	（- ラベル）の形式の式が用いられています。
	［対処方法］	式を確認してください。
E0550205	［メッセージ］	（- ラベル - ラベル）の形式の式が用いられています。
	［対処方法］	式を確認してください。
E0550206	［メッセージ］	（ラベル + ラベル）の形式の式が用いられています。
	［対処方法］	式を確認してください。
E0550207	［メッセージ］	異なる形式のラベル参照（#label，label，および $label）の間に演算が指定されています。
	［対処方法］	式を確認してください。
E0550208	［メッセージ］	異なるセクションに属するラベル間に演算が指定されています。
	［対処方法］	式を確認してください。
E0550209	［メッセージ］	ラベル同士の演算は同一ファイル内に定義してください。
	［対処方法］	式を確認してください。
E0550210	［メッセージ］	+,- 以外の演算が用いられています。
	［対処方法］	式を確認してください。
E0550211	［メッセージ］	内部的に用いている浮動小数点演算ライブラリの関数 *function* においてエラーが発生しました。
	［対処方法］	演算結果を確認してください。
E0550212	［メッセージ］	指定されたシンボルはすでにラベル *label* として定義されています。
	［対処方法］	シンボル名を確認してください。

E0550213	［メッセージ］	ラベル *identifier* が複数回定義されています。
	［対処方法］	ラベル名を確認してください。
E0550214	［メッセージ］	*identifier* が複数回定義されています。
	［対処方法］	ラベル名を確認してください。
E0550215	［メッセージ］	オペランドに異なるアクセス幅の内部周辺 I/O レジスタを指定しています。
	［対処方法］	内部周辺 I/O レジスタを確認してください。
E0550216	［メッセージ］	オペランドに指定した内部周辺 I/O レジスタは読み出し禁止です。
	［対処方法］	内部周辺 I/O レジスタを確認してください。
E0550217	［メッセージ］	オペランドに内部周辺 I/O レジスタのフラグのビットを指定することはできません。
	［対処方法］	内部周辺 I/O レジスタを確認してください。
E0550218	［メッセージ］	オペランドに指定した内部周辺 I/O レジスタは書き込み禁止です。
	［対処方法］	内部周辺 I/O レジスタを確認してください。
E0550219	［メッセージ］	ビット操作命令で指定したビット位置が矛盾しています。
	［対処方法］	ビット位置を確認してください。
E0550220	［メッセージ］	名前に予約語 *identifier* が用いられています。
	［対処方法］	オペランドを確認してください。
E0550221	［メッセージ］	分岐命令に対し、（ラベル - ラベル）の形式の式が指定されています。
	［対処方法］	式を確認してください。
E0550222	［メッセージ］	オペランドにラベルを指定することのできない命令に対し、ラベルが指定されています。
	［対処方法］	オペランドを確認してください。
E0550223	［メッセージ］	setf 命令、またはシフト命令に対し、ラベルが指定されています。
	［対処方法］	オペランドを確認してください。
E0550224	［メッセージ］	jmp 命令に対し、絶対アドレス参照（*string*）以外のものが指定されています。
	［対処方法］	オペランドを確認してください。
E0550225	［メッセージ］	式の評価結果が負になりました。
	［対処方法］	式を確認してください。
E0550226	［メッセージ］	奇数のディスプレースメントが指定されています。
	［対処方法］	ディスプレースメントを確認してください。
E0550227	［メッセージ］	演算子に、イミーディエト、ラベル、またはシンボル以外のものが指定されています。
	［対処方法］	オペランドを確認してください。
E0550228	［メッセージ］	レジスタ以外のものが指定されています。
	［対処方法］	オペランドを確認してください。
E0550229	［メッセージ］	ベース・レジスタを指定する必要があります。
	［対処方法］	オペランドを確認してください。

E0550230	［メッセージ］	ディスプレースメントとして指定された値が指定可能な値の範囲を越えています。
	［対処方法］	ディスプレースメントを確認してください。
E0550231	［メッセージ］	イミーディエトとして指定された値が指定可能な値の範囲を越えています。
	［対処方法］	イミーディエトを確認してください。
E0550232	［メッセージ］	.local 疑似命令に指定されたパラメータが不正です。
	［対処方法］	パラメータを確認してください。
E0550233	［メッセージ］	.local 疑似命令に指定されたパラメータがシンボルではありません。
	［対処方法］	パラメータを確認してください。
E0550234	［メッセージ］	.macro 疑似命令に指定されたパラメータが不正です。
	［対処方法］	パラメータを確認してください。
E0550235	［メッセージ］	.macro 疑似命令に定義されたマクロ名が不正です。
	［対処方法］	マクロ名を確認してください。
E0550236	［メッセージ］	マクロ呼び出しに指定された引数が不正です。
	［対処方法］	パラメータを確認してください。
E0550237	［メッセージ］	.irp 疑似命令に指定された引数が不正です。
	［対処方法］	引数を確認してください。
E0550238	［メッセージ］	.irp 疑似命令に指定されたパラメータが不正です。
	［対処方法］	パラメータを確認してください。
E0550239	［メッセージ］	*デバイス名*指定時には、ソース・オペランドに r0 を指定することはできません。
	［対処方法］	オペランドを確認してください。
E0550240	［メッセージ］	*デバイス名*指定時には、デスティネーション・オペランドに r0 を指定することはできません。
	［対処方法］	オペランドを確認してください。
E0550241	［メッセージ］	pushm/popm 命令に指定されたレジスタ数が多すぎます。
	［対処方法］	オペランドを確認してください。
E0550242	［メッセージ］	ラベルはすでに定義されています。(*section*)
	［説明］	指定したラベルはすでに *section* セクションに定義されています。
	［対処方法］	ラベルを確認してください。
E0550244	［メッセージ］	.org 疑似命令において値（*value*）の指定に誤りがあります。
	［対処方法］	値を確認してください。
E0550245	［メッセージ］	予約語を用いることのできない場所において予約語 *identifier* が用いられています。
	［対処方法］	記述を確認してください。
E0550246	［メッセージ］	セクション中に記述することのできない命令が記述されています。
	［対処方法］	記述を確認してください。
E0550247	［メッセージ］	サイズの指定に誤りがあります。
	［対処方法］	指定を確認してください。

E0550248	［メッセージ］	シンボル *symbol* に対し ‘$’、または ‘#’ が指定されています。
	［対処方法］	シンボルを確認してください。
E0550249	［メッセージ］	構成に誤りがあります。
	［対処方法］	記述を確認してください。
E0550250	［メッセージ］	*string* の構成に誤りがあります。
	［対処方法］	記述を確認してください。
E0550251	［メッセージ］	指定された ID 値が不正です。
	［説明］	指定された ID 値が不正です。整数を指定してください。
	［対処方法］	ID 値を確認してください。
E0550252	［メッセージ］	指定された ID 値は、シンボル名 "*identifier*" としてすでに予約されています。
	［対処方法］	ID 値を確認してください。
E0550253	［メッセージ］	指定されたシンボル名 "*identifier*" は、異なる ID 値ですでに予約されています。
	［対処方法］	ID 値を確認してください。
E0550254	［メッセージ］	$EXT_FUNC 疑似命令を用いて指定されたシンボル "*identifier*" は、分岐命令以外で参照することはできません。
	［対処方法］	記述を確認してください。
E0550255	［メッセージ］	I/O レジスタ名 "*I/O register*" に指定したビット番号が間違っています。
	［対処方法］	ビット番号を確認してください。
E0550258	［メッセージ］	オペランドに指定した内部周辺 I/O レジスタ "*I/O register*" はアクセス禁止です。
	［対処方法］	オペランドを確認してください。
E0550259	［メッセージ］	mul/mulu 命令のディスティネーション・レジスタにはアセンブラ予約レジスタ (r1) を指定できません。
	［対処方法］	ディスティネーション・レジスタを確認してください。
E0550260	［メッセージ］	トークンの長さが限界を越えています。
	［説明］	トークンの長さが限界を越えています。限界値は 4,294,967,294 です。
	［対処方法］	トークンの長さを確認してください。
E0550261	［メッセージ］	指定された条件コードが不正です。
	［説明］	【RH850】指定された条件コードが不正です。adf.sbf 命令【V850E2】の条件コードに 0xd は指定できません。
	［対処方法］	条件コードを確認してください。
E0550262	［メッセージ］	*デバイス名*指定時には、デスティネーション・オペランドに r0 を指定することはできません。
	［対処方法］	オペランドを確認してください。
E0550263	［メッセージ］	共通部指定時には、オペランドに $label,%label のラベル参照を指定することはできません。
	［対処方法］	オペランドを確認してください。
E0550264	［メッセージ］	セクション定義疑似命令において共通部で指定できないセクションの種類を指定しています。
	［対処方法］	セクションの種類を確認してください。

E0550265	[メッセージ]	指定できないレジスタ (r0-r7, r16-r31) が指定されています。
	[対処方法]	汎用レジスタとして指定できるのは，r8-r15 のいずれかです。オペランドを確認してください。
E0550266	[メッセージ]	指定できないベクトル・レジスタ（vr16-vr31）が指定されています。
	[対処方法]	ベクトル・レジスタとして指定できるのは，vr0-vr15 です。オペランドを確認してください。
E0550267	[メッセージ]	4 の倍数以外のディスプレースメントが指定されています。
	[対処方法]	ディスプレースメントを確認してください。
E0550268	[メッセージ]	指定できないベクトル・レジスタ（vr0-vr23）が指定されています。
	[説明]	ベクトル・レジスタとして指定できるのは，vr24-vr31 です。
	[対処方法]	オペランドを確認してください。
E0550269	[メッセージ]	RH850 " *コア名* " コア指定時には記述することのできない命令が記述されています。
	[説明]	指定されたコアでサポートされてない命令が記述されています。
	[対処方法]	命令のニモニックを確認してください。
E0550601	[メッセージ]	" *文字列* " オプションで指定された " *パス名* " はフォルダです。入力ファイルを指定してください。
E0550602	[メッセージ]	" *文字列* " オプションで指定されたファイル " *ファイル名* " が見つかりません。
	[対処方法]	ファイルが存在するか確認してください。
E0550603	[メッセージ]	" *文字列* " オプションで指定された " *パス名* " はフォルダです。出力ファイルを指定してください。
E0550604	[メッセージ]	" *文字列* " オプションで指定された出力先フォルダ " *フォルダ名* " が見つかりません。
E0550607	[メッセージ]	" *文字列* " オプションで指定された " *パス名* " が見つかりません。
	[説明]	" *文字列* " オプションで指定された " *パス名* "（ファイル名，またはフォルダ名）が見つかりません。
E0550608	[メッセージ]	" *文字列* " は認識されないオプションです。
E0550609	[メッセージ]	" *文字列* " オプションに引数は指定できません。
E0550610	[メッセージ]	" *文字列* " オプションに引数を指定してください。
E0550611	[メッセージ]	" *文字列* " オプションに引数は指定できません。
E0550612	[メッセージ]	" *文字列* " オプションに引数を指定してください。
	[説明]	" *文字列* " オプションは引数が必要です。
	[対処方法]	引数を指定してください。
E0550613	[メッセージ]	" *文字列* " オプションに指定された引数が不正です。
E0550617	[メッセージ]	" *文字列* " オプションに指定された引数が不正です。
E0550618	[メッセージ]	" *文字列* " オプションにはシンボルを指定してください。
E0550620	[メッセージ]	" *文字列* " オプションには *値 1* ～ *値 2* を指定してください。
E0550621	[メッセージ]	"-o" オプションと " *文字列* " オプションを同時に指定したとき、複数ソース・ファイルを入力できません。
E0550622	[メッセージ]	" *文字列* " オプションにはオブジェクト・ファイルを指定してください。

E0550624	[メッセージ]	"-Xcommon" オプションを指定してください。
E0550625	[メッセージ]	デバイス・ファイルが見つかりません。
E0550626	[メッセージ]	デバイス・ファイル " *ファイル名* " の読み込みに失敗しました。
E0550629	[メッセージ]	コマンド・ファイル " *ファイル名* " が複数回読まれています。
E0550630	[メッセージ]	コマンド・ファイル " *ファイル名* " が読み込めません。
E0550631	[メッセージ]	コマンド・ファイル " *ファイル名* " の構文が認識できません。
E0550632	[メッセージ]	テンポラリ・フォルダを作成できません。
E0550634	[メッセージ]	指定された入力ファイル " *ファイル名* " が見つかりません。
E0550635	[メッセージ]	指定された入力ファイル " *パス名* " はフォルダです。
E0550636	[メッセージ]	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0550642	[メッセージ]	" *文字列* " オプションには入力ファイルを複数指定できません。
E0550645	[メッセージ]	" *文字列 1* " オプションで指定された " *文字列 2* " は使用できません。
E0550648	[メッセージ]	ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。
	[対処方法]	異なるファイル名を指定してください。
E0550650	[メッセージ]	" *文字列 1* " オプションと " *文字列 2* " オプションが矛盾しています。
E0550652	[メッセージ]	" *文字列 1* " オプションには " *文字列 2* " オプションが必要です。
E0550654	[メッセージ]	CC-RH を構成するファイル " *ファイル名* " が見つかりません。再インストールしてください。
E0550661	[メッセージ]	ファイル " *ファイル名* " の削除に失敗しました。
E0550701	[メッセージ]	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0551200	[メッセージ]	アセンブリ・ソースの記述に誤りがあります。
	[対処方法]	アセンブリ・ソースを確認してください。
E0551201	[メッセージ]	カンマがありません。
	[対処方法]	カンマが抜けていないか確認してください。
E0551202	[メッセージ]	レジスタの記述に誤りがあります。
	[説明]	オペランドに指定できないレジスタの記述が含まれています。
	[対処方法]	オペランドに指定可能なレジスタを確認してください。
E0551203	[メッセージ]	リロケータブル項は記述できません。
	[説明]	リロケータブル項が許されていない箇所に記述されています。
	[対処方法]	該当箇所の記述形式を確認してください。
E0551204	[メッセージ]	オペランドの記述に誤りがあります。
	[説明]	オペランドに指定できない記述が含まれています。
	[対処方法]	オペランドに指定可能な形式を確認してください。
E0551205	[メッセージ]	文字列の記述に誤りがあります。
	[対処方法]	文字列の記述に誤りがないか確認してください。

E0551206	[メッセージ]	"$" は記述できません。
	[説明]	"$" が記述できない箇所に記述されています。
	[対処方法]	許されていない箇所に "$" を記述していないか確認してください。
E0551207	[メッセージ]	"*string*" は記述できません。
	[対処方法]	該当箇所の記述を確認ください。
E0551208	[メッセージ]	演算に誤りがあります ("op")。
	[説明]	"op" 演算の記述に誤りがあります。
	[対処方法]	"op" 演算の記述を確認してください。
E0551209	[対処方法]	bit 位置指定子の第 1 項に誤りがあります。
	[メッセージ]	bit 位置指定子の第 1 項の記述を確認してください。
E0551210	[メッセージ]	bit 参照後に分離演算はできません。
	[対処方法]	分離演算子は bit 参照の第 1 項に適用してください。
E0551211	[メッセージ]	bit シンボル以外は記述できません。
	[対処方法]	bit シンボル以外が記述されていないか確認してください。
E0551212	[メッセージ]	bit 位置指定子の記述に誤りがあります。
	[対処方法]	bit 位置指定子の記述を確認してください。
E0551213	[メッセージ]	かっこの不整合、または演算子の対象となる式がありません。
	[説明]	右かっこがないか，または演算子の対象となる式がありません。
	[対処方法]	かっこの対応が正しくとれているか，また演算の対象となる式があるかを確認してください。
E0551214	[メッセージ]	"op" 演算子の記述に誤りがあります。
	[対処方法]	op 演算子の記述形式を確認してください。
E0551215	[メッセージ]	ラベルの記述に誤りがあります。
	[対処方法]	ラベルの記述を確認してください。
E0551216	[メッセージ]	(ラベル - ラベル) 形式の式が用いられています。
	[説明]	(ラベル - ラベル) 形式の式が記述できない箇所に記述されています。
	[対処方法]	式を確認してください。
E0551217	[メッセージ]	(ラベル + ラベル) 形式の式が用いられています。
	[説明]	(ラベル + ラベル) 形式の式を記述することはできません。
	[対処方法]	式を確認してください。
E0551218	[メッセージ]	(- ラベル) 形式の式が用いられています。
	[対処方法]	式を確認してください。
E0551219	[メッセージ]	ラベルの演算、または参照が不正です。
	[対処方法]	ラベルの演算，または参照を確認してください。
E0551220	[メッセージ]	未定義シンボルは記述できません。
	[説明]	未定義シンボルが記述できない箇所に記述されています。
	[対処方法]	シンボルの定義を確認してください。

E0551221	[メッセージ]	セクション名は記述できません。
	[説明]	セクション名が記述できない箇所に記述されています。
	[対処方法]	指定可能な記述を確認してください。
E0551222	[メッセージ]	文字の読み込みに失敗しました。
	[対処方法]	記述を確認してください。
E0551223	[メッセージ]	シングルクォーテーションの記述を確認してください。
	[説明]	シングルクォーテーション (') が閉じられていません。
	[対処方法]	シングルクォーテーションが閉じられているか確認してください。
E0551224	[メッセージ]	文字列の読み取りに失敗しました。
	[対処方法]	記述を確認してください。
E0551225	[メッセージ]	ダブルクォーテーションの記述を確認してください。
	[説明]	ダブルクォーテーション (") が閉じられていません。
	[対処方法]	ダブルクォーテーションが閉じられているか確認してください。
E0551226	[メッセージ]	式の途中に文字列の記述がありました。
E0551227	[メッセージ]	'?' は英数字として扱いません。
	[説明]	'?' は英数字として扱いません。シンボル名として使用することはできません。
E0551228	[メッセージ]	-base_number で指定された進数指定と一致しません。
E0551229	[メッセージ]	2 進数の表現に誤りがあります。
	[対処方法]	2 進数の表記が正しいか確認してください。
E0551230	[説明]	8 進数の表現に誤りがあります。
	[メッセージ]	8 進数の表記が正しいか確認してください。
E0551231	[対処方法]	10 進数の表現に誤りがあります。
	[メッセージ]	10 進数の表記が正しいか確認してください。
E0551232	[メッセージ]	16 進数の表現に誤りがあります。
	[対処方法]	16 進数の表記が正しいか確認してください。
E0551233	[メッセージ]	オペランド数が多すぎます。
	[対処方法]	正しい数のオペランドを指定してください。
E0551234	[メッセージ]	右ブラケットがありません。
E0551235	[メッセージ]	ネストできない演算子です。
E0551236	[メッセージ]	チルダの記述に誤りがあります。
E0551301	[メッセージ]	bit 値は 0-7 の数値を記述してください。
E0551302	[メッセージ]	オペランドに指定した値が saddr 領域を超えています。
E0551303	[メッセージ]	オペランドに指定した値が sfr 領域を超えています。
E0551304	[メッセージ]	オペランドに指定した値が callt テーブル領域を超えています。
E0551305	[メッセージ]	オペランドに指定した値が 8bit 幅を超えています。
E0551306	[メッセージ]	オペランドに指定した値が 16bit 幅を超えています。

E0551307	[メッセージ]	オペランドに指定した値が 20bit 幅を超えています。
E0551308	[メッセージ]	オペランドに指定した値が 32bit 幅を超えています。
E0551309	[メッセージ]	奇数値が指定されました。
E0551310	[メッセージ]	"1" 以外の数値は記述できません。
E0551311	[メッセージ]	数値が 1-7 の範囲を超えています。
E0551312	[メッセージ]	数値が 1-15 の範囲を超えています。
E0551313	[メッセージ]	"*reg*" は記述できません。
	[説明]	ここには *reg* レジスタは記述できません。
	[対処方法]	記述可能なオペランドを確認してください。
E0551314	[対処方法]	HL 以外のレジスタは記述できません。
E0551315	[メッセージ]	ES 以外のレジスタは記述できません。
E0551316	[メッセージ]	SFR/ 制御レジスタの記述に誤りがあります。
E0552022	[メッセージ]	Symbol name is missing.
	[説明]	EQU 制御命令行にシンボル名が未記述です。
	[対処方法]	シンボル名を記述してください。
E0552023	[メッセージ]	Illegal directive command is used.
	[説明]	不正な制御命令を記述しています。
	[対処方法]	正しい制御命令に記述し直してください。
E0552024	[メッセージ]	No ';' at the top of comment.
	[説明]	コメント先頭に ";" が記述されていません。
	[対処方法]	コメントの先頭には，セミコロンを記述してください。ニモニック，またはオペランドの記述に誤りがないか確認してください。
E0552026	[メッセージ]	'CODE' section in big endian is not appropriate.
	[説明]	endian=big 時，絶対属性の CODE セクション開始アドレスに 4 の倍数以外の値を指定しています。
	[対処方法]	絶対属性の CODE セクション開始アドレスには 4 の倍数の値を指定してください。
E0552027	[メッセージ]	Illegal character code.
	[説明]	文字コードが正しくありません。
E0552028	[メッセージ]	Unrecognized character escape sequence.
	[説明]	認識できないエスケープシーケンスがあります。
E0552029	[メッセージ]	Invalid description in #pragma inline_asm function.
	[説明]	アセンブラ埋め込みインライン関数のアセンブリ言語に，使用できない記述があります。
E0552040	[メッセージ]	Include nesting over.
	[説明]	インクルードのネストレベルが深すぎます。
	[対処方法]	インクルードのネスとレベルが 9 以下になるように記述し直してください。

E0552041	[メッセージ]	Can't open include file 'XXXX'.
	[説明]	インクルード・ファイルをオープンできません。
	[対処方法]	インクルード・ファイル名を確認してください。インクルード・ファイルの格納ディレクトリを確認してください。
E0552042	[メッセージ]	Including the include file in itself.
	[説明]	インクルード・ファイル内で，自身をインクルードしています。
	[対処方法]	インクルード・ファイル名を確認して，記述し直してください。
E0552049	[メッセージ]	Invalid reserved word exist in operand.
	[説明]	オペランド中に予約語が記述されています。
	[対処方法]	予約語は，オペランドに記述できません。オペランドを記述し直してください。
E0552050	[メッセージ]	Operand value is not defined.
	[説明]	オペランドの値が未定義です。
	[対処方法]	オペランドには確定値を記述してください。
E0552051	[メッセージ]	'{' is missing.
	[説明]	'{' の記述がありません。
E0552052	[メッセージ]	Addressing mode specifier is not appropriate.
	[説明]	アドレッシングモード指定子の記述に間違いがあります。
	[対処方法]	アドレッシングモード指定子の記述方法を確認してください。
E0552053	[メッセージ]	Reserved word is missing.
	[説明]	予約語の記述がありません。
E0552054	[メッセージ]	']' is missing.
	[説明]	'] ' の記述がありません。
	[対処方法]	'[' に対応する ']' を記述してください。
E0552055	[メッセージ]	Right quote is missing.
	[説明]	右側の引用符がありません。
	[対処方法]	引用符を記述してください。
E0552056	[メッセージ]	The value is not constant.
	[説明]	値がアセンブル時確定値ではありません。
	[対処方法]	アセンブル時に確定するような，式，シンボル名又はラベル名を記述してください。
E0552057	[メッセージ]	Quote is missing.
	[説明]	文字列に対する引用符の記述がありません。
	[対処方法]	文字列は引用符で囲って記述してください。
E0552058	[メッセージ]	Illegal operand is used.
	[説明]	オペランドが間違っています。
	[対処方法]	オペランドの記述方法を確認して，記述し直してください。

E0552059	[メッセージ]	Operand number is not enough.
	[説明]	オペランドが不足しています。
	[対処方法]	オペランドの記述方法を確認して，記述し直してください。
E0552060	[メッセージ]	Too many macro nesting.
	[説明]	マクロのネスティングが多すぎます。
	[対処方法]	マクロのネスティングレベルを 65535 レベル以下にしてください。ソース記述を確認してください。
E0552061	[メッセージ]	Too many macro local label definition.
	[説明]	マクロ内ローカルラベルの定義が多すぎます。
	[対処方法]	マクロ内ローカルラベル数を 1 ファイル内に 65535 個以下にしてください。
E0552062	[メッセージ]	'.MACRO' is missing for '.ENDM'.
	[説明]	.ENDM に対する .MACRO がありません。
	[対処方法]	.ENDM の記述位置を確認してください。
E0552063	[対処方法]	'.MREPEAT' is missing for '.ENDR'.
	[メッセージ]	.ENDR に対する .MREPEAT がありません。
	[説明]	.ENDR の記述位置を確認してください。
E0552064	[メッセージ]	'.MACRO' or '.MREPEAT' is missing for '.EXITM'.
	[説明]	.EXITM に対する .MACRO 又は .MREPEAT がありません。
	[対処方法]	.EXITM の記述位置を確認してください。
E0552065	[メッセージ]	No macro name.
	[説明]	マクロ名がありません。
	[対処方法]	マクロ定義には，マクロ名を記述してください。
E0552066	[メッセージ]	Too many formal parameter.
	[説明]	マクロの仮引数の定義数が多すぎます。
	[対処方法]	マクロの仮引数の数を 80 以下にしてください。
E0552067	[メッセージ]	Illegal macro parameter.
	[説明]	マクロ引数に不正な記述があります。
	[対処方法]	マクロ引数の記述内容を確認してください。
E0552068	[メッセージ]	Source line is too long.
	[説明]	ソース行が長すぎます。
	[対処方法]	ソース行の記述内容を確認してください。
E0552069	[メッセージ]	'.MACRO' is missing for '.LOCAL'.
	[説明]	.LOCAL に対する .MACRO がありません。
	[対処方法]	.LOCAL の記述位置を確認してください。.LOCAL は，マクロブロック内にしか記述できません。

E0552070	[メッセージ]	No '.ENDM' statement.
	[説明]	.ENDM 記述がありません。
	[対処方法]	.ENDM の記述位置を確認してください。.ENDM を記述してください。
E0552071	[メッセージ]	No '.ENDR' statement.
	[説明]	.ENDR 記述がありません。
	[対処方法]	.ENDR の記述位置を確認してください。.ENDR を記述してください。
E0552072	[メッセージ]	')' is missing.
	[説明]	')' の記述がありません。
	[対処方法]	'(' に対応する ')' を記述してください。
E0552073	[メッセージ]	Operand expression is not completed.
	[説明]	オペランド記述に不足があります。
	[対処方法]	オペランドの記述方法を確認して，記述し直してください。
E0552074	[メッセージ]	Syntax error in expression.
	[説明]	式の記述に間違いがあります。
	[対処方法]	式の記述方法を確認して，記述し直してください。
E0552075	[メッセージ]	String value exist in expression.
	[説明]	式中に文字列式が記述されています。
	[対処方法]	式を記述し直してください。
E0552076	[メッセージ]	Division by zero.
	[説明]	除数 0 による除算が行われています。
	[対処方法]	式を記述し直してください。
E0552077	[メッセージ]	No '.END' statement.
	[説明]	.END の記述がありません。
	[対処方法]	ソースプログラムの最後の行に .END を記述してください。
E0552078	[メッセージ]	The specified address overlaps at ' *アドレス値* '.
	[説明]	指定された ' *アドレス値* ' はアドレス割付けが重複しています。C/C++ ソースの場合は ' *アドレス値* ' で複数のシンボルが重複しています。
	[対処方法]	.ORG，.OFFSET の指定内容を見直してください。' *アドレス値* ' に割り当たっているシンボルを確認してください。
E0552080	[メッセージ]	'.IF' is missing for '.ELSE'.
	[説明]	.ELSE に対する .IF がありません。
	[対処方法]	.ELSE の記述位置を確認してください。
E0552081	[メッセージ]	'.IF' is missing for '.ELIF'.
	[説明]	.ELIF に対する .IF がありません。
	[対処方法]	.ELIF の記述位置を確認してください。

E0552082	[メッセージ]	'.IF' is missing for '.ENDIF'.
	[説明]	.ENDIF に対する .IF がありません。
	[対処方法]	.ENDIF の記述位置を確認してください。
E0552083	[メッセージ]	Too many nesting level of condition assemble.
	[説明]	条件アセンブルのネスティングが多すぎます。
	[対処方法]	条件アセンブルの記述を確認してください。
E0552084	[メッセージ]	No '.ENDIF' statement.
	[説明]	ソース・ファイル内に IF 文に対応した ENDIF がありません。
	[対処方法]	ソースの記述を確認してください。
E0552088	[メッセージ]	Can't open '.ASSERT' message file 'XXXX'.
	[説明]	.ASSERT の出力ファイルをオープンできません。
	[対処方法]	ファイル名を確認してください。
E0552089	[メッセージ]	Can't write '.ASSERT' message file 'XXXX'.
	[説明]	.ASSERT の出力ファイルに書き込みできません。
	[対処方法]	ファイルのパーミッションを確認してください。
E0552090	[メッセージ]	Too many temporary label.
	[説明]	テンポラリラベルの個数が多すぎます。
	[対処方法]	テンポラリラベルをラベル名に置き換えて記述してください。
E0552091	[メッセージ]	Temporary label is undefined.
	[説明]	テンポラリラベルが未定義です。
	[対処方法]	テンポラリラベルの定義を行ってください。
E0552100	[メッセージ]	Value is out of range.
	[説明]	値が範囲外です。
	[対処方法]	レジスタなどのビット長に合った値を記述してください。
E0552112	[メッセージ]	Symbol is missing.
	[説明]	シンボルの記述がありません。
	[対処方法]	シンボル名を記述してください。
E0552113	[メッセージ]	Symbol definition is not appropriate.
	[説明]	シンボルの定義に間違いがあります。
	[対処方法]	シンボル定義方法を確認して記述し直してください。
E0552114	[メッセージ]	Symbol has already defined as another type.
	[説明]	シンボルは既に同一名で異なる制御命令で定義されています。
	[対処方法]	シンボル名を変更してください。
E0552115	[メッセージ]	Symbol has already defined as the same type.
	[説明]	シンボルは，すでに定義されています。
	[対処方法]	シンボル名を変更してください。

E0552116	[メッセージ]	Symbol is multiple defined.
	[説明]	シンボルが二重定義です。マクロ名とほかの名前が重複しています。
	[対処方法]	シンボル名を変更してください。
E0552117	[メッセージ]	Invalid label definition.
	[説明]	無効なラベル記述をしています。
	[対処方法]	ラベル定義を記述し直してください。
E0552118	[メッセージ]	Invalid symbol definition.
	[説明]	無効なシンボル記述をしています。
	[対処方法]	シンボルの定義を記述し直してください。
E0552119	[メッセージ]	Reserved word is used as label or symbol.
	[説明]	予約語をラベル，またはシンボルに用いています。
	[対処方法]	ラベル，またはシンボル名を記述し直してください。
E0552130	[メッセージ]	No '.SECTION' statement.
	[説明]	'.SECTION ' の記述がありません。
	[対処方法]	ソースプログラムには，必ず 1 つ以上の .SECTION を記述してください。
E0552131	[メッセージ]	Section type is not appropriate.
	[説明]	セクション属性の記述が間違っています。
	[対処方法]	セクション属性を記述し直してください。
E0552132	[メッセージ]	Section has already determined as attribute.
	[説明]	セクションは既に相対属性に確定しています。制御命令 ".ORG" は記述できません。
	[対処方法]	セクションの属性を確認してください。
E0552133	[メッセージ]	Section attribute is not defined.
	[説明]	セクションの属性が未定義です。このセクション内では制御命令 ".ALIGN" は記述できません。
	[対処方法]	制御命令 ".ALIGN" は，絶対アドレス属性セクション又は ALIGN 指定のある相対アドレス属性セクション内に記述してください。
E0552134	[メッセージ]	Section name is missing.
	[説明]	セクション名がありません。
	[対処方法]	オペランドにセクション名を記述してください。
E0552135	[メッセージ]	'ALIGN' is multiple specified in '.SECTION'.
	[説明]	.SECTION 定義行に複数の 'ALIGN ' 指定があります。
	[対処方法]	余分な 'ALIGN' 指定を削除してください。
E0552136	[メッセージ]	Section type is multiple specified.
	[説明]	セクション定義行には，"CODE"，"DATA"，"ROMDATA" の指定は 1 つだけ記述してください。

E0552137	[メッセージ]	Too many operand.
	[説明]	オペランドが余分にあります。
	[対処方法]	オペランドの記述内容を確認してください。
E0560601	[メッセージ]	ディレクティブの終わりには ';' が必要とされます。
E0560602	[メッセージ]	各領域の終わりには '}' が必要とされます。
E0560603	[メッセージ]	ディレクティブは名前 (セグメント名 / セクション名 / シンボル名) ではじめてください。
E0560604	[メッセージ]	セグメント・ディレクティブはセクション名ではじめてください。
E0560605	[メッセージ]	ディレクティブの始まりの名前の後ろには ':','=' または '@' が必要とされます。
E0560606	[メッセージ]	出力セクション名の後ろには '=' が必要とされます。
E0560607	[メッセージ]	'{' に対応する '}' の数が多すぎます。
E0560608	[メッセージ]	リンク・ディレクティブに不正な文字 (*number*) が存在しています。
E0560609	[メッセージ]	"*stirng*" には有効なパラメータが必要です。
E0560610	[メッセージ]	セグメント・ディレクティブにおいて指定することができない文字列が使用されています。
E0560611	[メッセージ]	セクション・ディレクティブにおいて指定することができない文字列が使用されています。
E0560612	[メッセージ]	シンボル・ディレクティブにおいて指定することができない文字列が使用されています。
E0560613	[メッセージ]	ファイル名を指定する部分に "*stirng*" を指定することはできません。
E0560614	[メッセージ]	セグメント名を指定する部分に "*stirng*" を指定することはできません。
E0560615	[メッセージ]	"*stirng*" が、セグメント "*segment*" に対し、同じセグメント・ディレクティブまたは別のセグメント・ディレクティブにおいて複数回指定されてます。
E0560616	[メッセージ]	"*stirng*" が、セクション "*section*" に対し、同じセクション・ディレクティブまたは別のセクション・ディレクティブにおいて複数回指定されてます。
E0560617	[メッセージ]	"*stirng*" が、シンボル "*symbol*" に対し、同じシンボル・ディレクティブまたは別のシンボル・ディレクティブにおいて複数回指定されてます。
E0560618	[メッセージ]	セグメント "*segment*" はすでに定義されています。
E0560619	[メッセージ]	セクション "*section*" は *number* 行目においてすでに定義されています。
E0560620	[メッセージ]	シンボル "*symbol*" は *number* 行目においてすでに定義されています。
E0560621	[メッセージ]	セグメント・タイプとして指定することができない "*string*" が指定されています。
E0560622	[メッセージ]	セクション・タイプとして指定することができない "*string*" が指定されています。
E0560623	[メッセージ]	セクション属性として指定することができない '*character*' が指定されています。
E0560624	[メッセージ]	セグメント・タイプに LOAD を指定していないメモリ・ディレクティブにおいて "*stirng*" を指定することはできません。
E0560637	[メッセージ]	シンボル種別として指定することができない "*string*" が指定されています。
E0560638	[メッセージ]	シンボル種別 "*string*" が、同じディレクティブまたは別のディレクティブにおいて複数回指定されています。
E0560641	[メッセージ]	セクション "*section*" のセクション属性 *attribute* と本セクションの割り付けが指示されているセグメント "*segment*" のセグメント属性が一致しません。

E0560660	[メッセージ]	S-JIS コード (*number1*, *number2*) が不正です。
E0562000	[メッセージ]	Invalid option : " *オプション*"
	[説明]	" *オプション*" はサポートしていません。
E0562001	[メッセージ]	Option " *オプション*" cannot be specified on command line
	[説明]	" *オプション*" はコマンド・ライン上では指定できません。
	[対処方法]	サブコマンド・ファイル内で指定してください。
E0562002	[メッセージ]	Input option cannot be specified on command line
	[説明]	コマンド・ライン上で input オプションを指定しました。
	[対処方法]	コマンド・ライン上での入力ファイル指定は input オプションなしで指定してください。
E0562003	[メッセージ]	Subcommand option cannot be specified in subcommand file
	[説明]	サブコマンド・ファイル内に -subcommand オプションを指定しました。 -subcommand オプションはネストできません。
E0562004	[メッセージ]	Option " *オプション1*" cannot be combined with option " *オプション2*"
	[説明]	" *オプション1*" と " *オプション2*" は同時に指定できません。
E0562005	[メッセージ]	Option " *オプション*" cannot be specified while processing " *プロセス* "
	[説明]	" *プロセス* " 処理に対して " *オプション*" は指定できません。
E0562006	[メッセージ]	Option " *オプション1*" is ineffective without option " *オプション2*"
	[説明]	" *オプション1*" は " *オプション2*" が必要です。
E0562010	[メッセージ]	Option " *オプション*" requires parameter
	[説明]	" *オプション*" はパラメータ指定が必要です。
E0562011	[メッセージ]	Invalid parameter specified in option " *オプション*" : " *パラメータ* "
	[説明]	" *オプション*" で無効なパラメータを指定しました。
E0562012	[メッセージ]	Invalid number specified in option " *オプション*" : " *値* "
	[説明]	" *オプション*" 指定で無効な値を指定しました。
	[対処方法]	値の範囲を確認してください。
E0562013	[メッセージ]	Invalid address value specified in option " *オプション*" : " *アドレス* "
	[説明]	" *オプション*" で指定した " *アドレス* " は無効な値です。
	[対処方法]	0 ～ FFFFFFFF の間の 16 進数で指定してください。
E0562014	[メッセージ]	Illegal symbol/section name specified in " *オプション*" : " *名前* "
	[説明]	" *オプション*" で指定したセクションまたはシンボル名に不正文字が使用されています。
E0562016	[メッセージ]	Invalid alignment value specified in option " *オプション*" : " *整列条件数* "
	[説明]	" *オプション*" で指定した " *整列条件数* " は無効な値です。
	[対処方法]	1，2，4，8，16，または 32 を指定してください。

E0562017	[メッセージ]	Cannot output " *セクション* " specified in option " *オプション* "
	[説明]	" *オプション* " で指定した " *セクション* " のコードの一部を出力できません。命令コードのエンディアンを変換したことにより，" *セクション* " 内命令コードの一部が非連続となりました。
	[対処方法]	非連続部分の命令コードが属しているセクションは，リンク・マップからセクションアドレスを 4 バイト境界で確認のうえ，出力するセクションがどのセクションとエンディアン変換を行っているか確認してください。
E0562020	[メッセージ]	Duplicate file specified in option " *オプション* " : " *ファイル* "
	[説明]	" *オプション* " 指定で同じファイルを二度指定しました。
E0562021	[メッセージ]	Duplicate symbol/section specified in option " *オプション* " : " *名前* "
	[説明]	" *オプション* " 指定で同じシンボル名またはセクション名を二度指定しました。
E0562022	[メッセージ]	Address ranges overlap in option " *オプション* " : " *アドレス範囲* "
	[説明]	" *オプション* " で指定した " *アドレス範囲* " が重複しています。
E0562100	[メッセージ]	Invalid address specified in cpu option : " *アドレス* "
	[説明]	-cpu オプションで cpu では指定できないアドレスを指定しました。
E0562101	[メッセージ]	Invalid address specified in option " *オプション* " : " *アドレス* "
	[説明]	" *オプション* " で指定した " *アドレス* " は cpu で指定できるアドレス範囲，または -cpu オプションで指定した範囲を越えました。
E0562110	[メッセージ]	Section size of second parameter in rom option is not 0 : " *セクション* "
	[説明]	-rom オプションの第 2 パラメータにサイズが 0 でない " *セクション* " を指定しました。
E0562111	[メッセージ]	Absolute section cannot be specified in " *オプション* " option : " *セクション* "
	[説明]	" *オプション* " で絶対アドレス・セクションを指定しました。
E0562112	[メッセージ]	" *セクション1* " and " *セクション2* " cannot mapped as ROM/RAM in " *ファイル* "
	[説明]	" *ファイル名* " で指定された " *セクション1* " と " *セクション2* " は ROM/RAM 対応となりません。
E0562113	[メッセージ]	Option "rom" and internal information in the file are conflicted
	[説明]	-rom オプションの指定と内部情報が矛盾しています。
E0562120	[メッセージ]	Library " *ファイル* " without module name specified as input file
	[説明]	入力ファイルとしてモジュール名なしのライブラリ・ファイルを指定しました。
E0562121	[メッセージ]	Input file is not library file : " *ファイル* (*モジュール*)"
	[説明]	入力ファイルで指定した " *ファイル* (*モジュール*)" はライブラリ・ファイルではありません。
E0562130	[メッセージ]	Cannot find file specified in option " *オプション* " : " *ファイル* "
	[説明]	" *オプション* " で指定したファイルが見つかりません。
E0562131	[メッセージ]	Cannot find module specified in option " *オプション* " : " *モジュール* "
	[説明]	" *オプション* " で指定したモジュールがありません。
E0562132	[メッセージ]	Cannot find " *名前* " specified in option " *オプション* "
	[説明]	" *オプション* " で指定したシンボルまたはセクションが存在しません。

E0562133	［メッセージ］	Cannot find defined symbol " *名前* " in option " *オプション* "
	［説明］	" *オプション* " で指定した外部定義シンボルが存在しません。
E0562134	［メッセージ］	Reserved section name " *セクション* "
	［説明］	" *セクション* " は，リンカで使用する予約名称です。
	［対処方法］	セクション名が正しいか確認してください。
E0562135	［メッセージ］	Interrupt number " *ベクタ番号* " has invalid interrupt jump address : " *シンボル* "
	［説明］	" *シンボル* " を割り込み関数として指定できません。
	［対処方法］	オプションとソース・ファイルの記述を確認してください。
E0562140	［メッセージ］	Symbol/section " *名前* " redefined in option " *オプション* "
	［説明］	" *オプション* " で指定したシンボル，セクションはすでに定義されています。
E0562141	［メッセージ］	Module " *モジュール* " redefined in option " *オプション* "
	［説明］	" *オプション* " で指定したモジュールはすでに登録されています。
E0562142	［メッセージ］	Interrupt number " *ベクタ番号* " of " *セクション* " has multiple definition
	［説明］	ベクタテーブル " *セクション* " の，ベクタ番号定義が複数入力されました。ベクタ番号には，ひとつのアドレスしか設定できません。
	［対処方法］	ソース・ファイルの記述を見直してください。
E0562200	［メッセージ］	Illegal object file : " *ファイル* "
	［説明］	ELF フォーマット以外を入力しました。
E0562201	［メッセージ］	Illegal library file : " *ファイル* "
	［説明］	" *ファイル* " はライブラリ・ファイルではありません。
E0562203	［メッセージ］	Illegal profile information file : " *ファイル* "
	［説明］	" *ファイル* " はプロファイル情報ファイルではありません。
E0562204	［メッセージ］	Unsupported device file : " *ファイル* "
	［説明］	" *ファイル* " をデバイスファイルとして読めません。
	［対処方法］	デバイス・ファイルを確認してください。
E0562210	［メッセージ］	Invalid input file type specified for option " *オプション* " : " *ファイル* (*種別*)"
	［説明］	" *オプション* " 指定時に処理できない " *ファイル* (*種別*)" を入力しました。
E0562211	［メッセージ］	Invalid input file type specified while processing " *プロセス* " : " *ファイル* (*種別*)"
	［説明］	" *プロセス* " 処理に対して処理できない " *ファイル* (*種別*)" を入力しました。
E0562212	［メッセージ］	" *オプション* " cannot be specified for inter-module optimization information in " *ファイル* "
	［説明］	" *ファイル* " 内にモジュール間最適化情報があるため，" *オプション* " オプションは使用できません。
	［対処方法］	コンパイル，アセンブル時に -goptimize オプションを使用しないでください。
E0562220	［メッセージ］	Illegal mode type " *モード種別* " in " *ファイル* "
	［説明］	異なる " *モード種別* " のファイルを入力しました。

E0562221	［メッセージ］	Section type mismatch : " *セクション*"
	［説明］	属性 (初期値有無) の異なる同名セクションを入力しました。
E0562223	［メッセージ］	Cpu type "*CPU 種別 1*" in " ファイル " is incompatible with "*CPU 種別 2*"
	［説明］	異なる CPU 種別を入力しました。 一部の仕様に互換性がないため，リンクしても動作が保証できません。
E0562224	［メッセージ］	Section type (relocation attribute) mismatch : " *セクション*"
	［説明］	再配置属性が異なる同名セクションを入力しました。
E0562300	［メッセージ］	Duplicate symbol " *シンボル* " in " *ファイル* "
	［説明］	" *シンボル* " は重複しています。
E0562301	［メッセージ］	Duplicate module " *モジュール* " in " *ファイル* "
	［説明］	" *モジュール* " は重複しています。
E0562310	［メッセージ］	Undefined external symbol " *シンボル* " referenced in " *ファイル* "
	［説明］	" *ファイル* " 内で未定義の " *シンボル* " を参照しています。
E0562311	［メッセージ］	Section " *セクション 1*" cannot refer to overlaid section : " *セクション 2*"-" *シンボル* "
	［説明］	同一アドレスを指定したオーバレイセクション間でシンボル参照がありました。
	［対処方法］	" *セクション 1*" と " *セクション 2*" を同じアドレスに割り付けないでください。
E0562320	［メッセージ］	Section address overflowed out of range : " *セクション* "
	［説明］	" *セクション* " のアドレスが使用可能なアドレス範囲を越えました。
E0562321	［メッセージ］	Section " *セクション 1*" overlaps section " *セクション 2*"
	［説明］	" *セクション 1*" と " *セクション 2*" のアドレスが重複しました。
	［対処方法］	start オプションのアドレス指定を変更してください。
E0562323	［メッセージ］	Section " *セクション 1(アドレス範囲)*" overlaps with section " *セクション 2(アドレス範囲)*" in physical space
	［説明］	物理メモリの配置上で，" *セクション 1*" と " *セクション 2*" が重複しています。
	［対処方法］	各セクションの配置アドレスを見直してください。 *< アドレス範囲 >* : *< セクションの開始アドレス >*-*< セクションの終端アドレス >*
E0562324	［メッセージ］	Section " *セクション* " in " *ファイル* " conflicts
	［説明］	" *セクション* " が存在するオブジェクト・ファイルを複数入力しました。
E0562330	［メッセージ］	Relocation size overflow : " *ファイル* "-" *セクション* "-" *オフセット* "
	［説明］	リロケーション演算結果がリロケーションサイズを越えました。分岐先が届かない，特定のアドレスに配置しなければならないシンボルを参照しているなどが考えられます。
	［対処方法］	アセンブル・リストで，" *セクション* " の " *オフセット* " 位置の参照シンボルが正しい位置に配置されているか確認してください。
E0562331	［メッセージ］	Division by zero in relocation value calculation : " *ファイル* "-" *セクション* "-" *オフセット* "
	［説明］	リロケーション演算に 0 除算が発生しました。
	［対処方法］	コンパイル，アセンブル・リストで，" *セクション* " の " *オフセット* " 位置の演算に問題がないか確認してください。

E0562332	［メッセージ］	Relocation value is odd number : " *ファイル*"-" *セクション*"-" *オフセット*"
	［説明］	リロケーション演算結果が奇数になりました。
	［対処方法］	コンパイル，アセンブル・リストで，" *セクション*" の " *オフセット*" 位置の演算に問題がないか確認してください。
E0562340	［メッセージ］	Symbol name " *ファイル*"-" *セクション*"-" *シンボル* ..." is too long
	［説明］	" *セクション*" 内の " *シンボル*" の文字数がアセンブラの翻訳限界を越えました。
	［対処方法］	シンボル・アドレス・ファイルを出力する場合は，アセンブラの翻訳限界文字数以下になるようなシンボル名としてください。
E0562402	［メッセージ］	Number of register parameter conflicts with that in another file : " *関数*"
	［説明］	" *関数*" は複数のファイルで異なるレジスタ・パラメータ数が指定されています。
E0562403	［メッセージ］	Fast interrupt register in " *ファイル*" conflicts with that in another file
	［説明］	" *ファイル*" 内で指定した高速割り込み用汎用レジスタ番号が，他ファイルと統一されていません。
	［対処方法］	高速割り込み用汎用レジスタ番号を他ファイルに合わせて，再度コンパイルして下さい。
E0562404	［メッセージ］	Base register " *ベースレジスタ種別*" in " *ファイル*" conflicts with that in another file
	［説明］	" *ファイル*" 内で指定した " *ベースレジスタ種別*" 用のレジスタ番号が，他ファイルと統一されていません。
	［対処方法］	ベース・レジスタ番号を他ファイルに合わせて，再度コンパイルして下さい。
E0562405	［メッセージ］	Option " *コンパイル・オプション*" conflicts with that in other files
	［説明］	" *コンパイル・オプション*" の指定が入力ファイル間で統一されていません。
	［対処方法］	コンパイル・オプションを見直してください。
E0562406	［メッセージ］	General-purpose register " *レジスタ*" in " *ファイル*" conflicts with that in another file(" *モード*")
	［説明］	" *ファイル*" 内で指定した汎用レジスタ " *レジスタ*" の使用方法が，他ファイルと統一されていません。
	［対処方法］	コンパイル時のオプションを確認してください。
E0562407	［メッセージ］	Handler stack pointer " *レジスタ*" in " *ファイル*" conflicts with that in another file(" *モード*")
	［説明］	" *ファイル*" 内で指定したハンドラ・スタック・ポインタ " *レジスタ*" の使用方法が，他ファイルと統一されていません。
	［対処方法］	コンパイル時のオプションを確認してください。
E0562408	［メッセージ］	Register mode in " *ファイル*" conflicts with that in another file(" *モード*")
	［説明］	複数のファイルで異なるレジスタ・モードを指定されています。
	［対処方法］	コンパイル時のオプションを確認してください。

E0562410	［メッセージ］	Address value specified by map file differs from one after linkage as to " *シンボル* "
	［説明］	" *シンボル* " のアドレス値がコンパイル時に使用した外部シンボル割り付け情報ファイル内のアドレスとリンク後のアドレスで異なっています。
	［対処方法］	以下の (1) ～ (3) を確認してください。 (1) コンパイル時の map オプション指定前後でプログラムを変更している場合は，プログラムの変更をやめてください。 (2) rlink の最適化によって，コンパイル時の map オプション指定前後のシンボル並び順が変わることがあります。コンパイル時 map オプションを無効にするか，rlink の最適化オプションを無効にしてください。 (3) tbr オプションまたは #pragma tbr 使用時，コンパイラの最適化によって，コンパイル時の map オプション指定後のシンボルが削除されることがあります。コンパイル時 map オプションを無効にするか，tbr オプションまたは #pragma tbr を無効にしてください。
E0562411	［メッセージ］	Map file in " *ファイル* " conflicts with that in another file
	［説明］	入力ファイル間でコンパイル時に異なる外部シンボル割り付け情報ファイルを使用しています。
E0562412	［メッセージ］	Cannot open file : " *ファイル* "
	［説明］	" *ファイル* "(外部シンボル割り付け情報ファイル) がオープンできません。
	［対処方法］	ファイル名およびアクセス権が正しいか確認してください。
E0562413	［メッセージ］	Cannot close file : " *ファイル* "
	［説明］	" *ファイル* "(外部シンボル割り付け情報ファイル) がクローズできません。ディスク容量に空きがない可能性があります。
E0562414	［メッセージ］	Cannot read file : " *ファイル* "
	［説明］	" *ファイル* "(外部シンボル割り付け情報ファイル) が読みこめません。ディスク容量に空きがない可能性があります。
E0562415	［メッセージ］	Illegal map file : " *ファイル* "
	［説明］	" *ファイル* "(外部シンボル割り付け情報ファイル) のフォーマットが不正です。
	［対処方法］	ファイル名が正しいか確認してください。
E0562416	［メッセージ］	Order of functions specified by map file differs from one after linkage as to " *関数名* "
	［説明］	関数 " *関数名* " は，コンパイル時に使用した外部シンボル割り付け情報ファイル内の情報とリンク後の配置とで，ほかの関数との並び順が異なっています。関数内 static 変数のアドレスが，外部シンボル割り付け情報ファイルとリンク後の結果とで異なっている可能性があります。
E0562417	［メッセージ］	Map file is not the newest version: " *ファイル名* "
	［説明］	外部シンボル割り付け情報ファイルが最新バージョンではありません。
E0562420	［メッセージ］	" *ファイル1* " overlap address " *ファイル2* " : " *アドレス* "
	［説明］	" *ファイル1* " と " *ファイル2* " のアドレスが重複しています。
E0562500	［メッセージ］	Cannot find library file : " *ファイル* "
	［説明］	ライブラリとして指定した " *ファイル* " がありません。
E0572000	［メッセージ］	Invalid option : " *オプション* "
	［説明］	" *オプション* " はサポートしていません。

E0572200	［メッセージ］	Illegal object file : " *ファイル* "
	［説明］	ELF フォーマット以外を入力しました。
E0572500	［メッセージ］	Cannot find library file : " *ファイル* "
	［説明］	ライブラリとして指定した " *ファイル* " がありません。
E0572501	［メッセージ］	" *インスタンス* " has been referenced as both an explicit specialization and a generated instantiation
	［説明］	すでに定義が存在しているインスタンスに対して，インスタンス生成を要求しています。
	［対処方法］	" *インスタンス* " を使用しているファイルに対して，form=relocate でリロケータブル・ファイルを作成していないか確認してください。
E0572502	［メッセージ］	" *インスタンス* " assigned to " *ファイル 1*" and " *ファイル 2*"
	［説明］	" *ファイル 1*" と " *ファイル 2*" に " *インスタンス* " 定義が重複しています。
	［対処方法］	" *インスタンス* " を使用しているファイルに対して，form=relocate でリロケータブル・ファイルを作成していないか確認してください。
E0573005	［メッセージ］	Instantiation loop
	［説明］	インスタンス生成処理がループしています。入力ファイル名が別ファイルのファイル名と一致している可能性があります。
	［対処方法］	拡張子を除いたファイル名が一致しないようにファイル名を変更してください。
E0573007	［メッセージ］	Cannot create instantiation request file " *ファイル* "
	［説明］	インスタンス生成処理用の中間ファイルを作成できません。
	［対処方法］	オブジェクト作成フォルダ以下のアクセス権が正しいか確認してください。
E0573008	［メッセージ］	Cannot change to directory " *フォルダ* "
	［説明］	" *フォルダ* " に移動できません。
	［対処方法］	" *フォルダ* " が存在するか確認してください。
E0573009	［メッセージ］	File " *ファイル* " is read-only
	［説明］	" *ファイル* " は読み取り専用です。
	［対処方法］	アクセス権を変更してください。
E0573300	［メッセージ］	Cannot open file : " *ファイル* "
	［説明］	" *ファイル* " をオープンできません。
	［対処方法］	ファイル名およびアクセス権が正しいか，確認してください。
E0573303	［メッセージ］	Cannot read file : " *ファイル* "
	［説明］	" *ファイル* " を読めません。空ファイルを入力したか，ディスク容量に空きがない可能性があります。
E0573310	［メッセージ］	Cannot open temporary file
	［説明］	中間ファイルをオープンできません。HLNK_TMP 指定が正しいか確認してください。またはディスク容量に空きがない可能性があります。
E0573320	［メッセージ］	Memory overflow
	［説明］	最適化リンケージエディタが内部で使用するメモリが不足しています。
	［対処方法］	メモリを増やしてください。

E0573500	[メッセージ]	Bad instantiation request file -- instantiation assigned to more than one file
	[説明]	インスタンス生成処理用の中間ファイルに誤りがあります。
	[対処方法]	リンク対象ファイルを再コンパイルしてください。
E0573505	[メッセージ]	corrupted template information file or instantiation request file
	[説明]	テンプレート処理用中間ファイル，またはインスタンス生成処理用の中間ファイルのデータが誤っています。これらのファイルの編集はしないでください。
E0592001	[メッセージ]	複数の入力ファイルが指定されています。
	[対処方法]	複数のファイルをコンバートする際は，リスト・ファイルを使用してください。
E0592002	[メッセージ]	複数の出力ファイルが指定されています。
	[対処方法]	複数のファイルをコンバートする際は，リスト・ファイルを使用してください。
E0592003	[メッセージ]	複数のリスト・ファイルが指定されています。
	[対処方法]	1 つのリスト・ファイルにまとめてください。
E0592004	[メッセージ]	"*option*" オプションに指定された引数が不正です。
	[対処方法]	引数を確認してください。
E0592005	[メッセージ]	"*option*" オプションに引数は指定できません。
	[説明]	引数の指定できない "*option*" オプションに引数が指定されています。
E0592006	[メッセージ]	"*option*" オプションに引数を指定してください。
	[説明]	引数の必要な "*option*" オプションに引数が指定されてません。
E0592007	[メッセージ]	"*option*" オプションが複数回指定されています。
	[説明]	"*option*" オプションは，同時に 1 つしか指定できません。
E0592008	[メッセージ]	出力ファイルが指定されていません。
	[説明]	指定された入力ファイルに対応する出力ファイルが指定されていません。
E0592010	[メッセージ]	出力ファイル "*file*" のオープンに失敗しました。
E0592013	[メッセージ]	テンポラリ・ファイル "*file*" の削除に失敗しました。
E0592015	[メッセージ]	入力ファイル "*file*" のクローズに失敗しました。
E0592016	[メッセージ]	出力ファイル "*file*" の書き込みに失敗しました。
E0592018	[メッセージ]	リスト・ファイル "*file*" のオープンに失敗しました。
	[対処方法]	存在するリスト・ファイルが正しく指定されているか確認してください。
E0592019	[メッセージ]	リスト・ファイル "*file*" の構文が認識できません。
	[説明]	リスト・ファイル "*file*" の記述が正しくありません。
E0592020	[メッセージ]	リスト・ファイル "*file*" の読み込みに失敗しました。
E0592101	[メッセージ]	不明な文字がありました '*C*'。
	[説明]	変換前の C 言語ソース・ファイルに C 言語で許可されていない文字があるため変換できません。
	[対処方法]	C 言語ソース・ファイルを編集して，構文エラーを修正してください。

E0592102	［メッセージ］	*string* の書式が不正です。
	［説明］	変換前の C 言語ソース・ファイルで許可されていない文法があるため変換できません。
	［対処方法］	C 言語ソース・ファイルを編集して，構文エラーを修正してください。
E0592201	［メッセージ］	構文に誤りがあるため変換できません。
	［説明］	変換前のアセンブリ言語ソース・ファイルに構文の誤りがあるため変換できません。
	［対処方法］	アセンブリ言語ソース・ファイルを編集して，構文エラーを修正してください。
E0593002	［メッセージ］	“-Xsfg_size_tidata_byte” サイズが、“-Xsfg_size_tidata” サイズより大きいです。
	［対処方法］	“-Xsfg_size_tidata_byte” サイズを，“-Xsfg_size_tidata” サイズ以下に設定するか，“-Xsfg_size_tidata” サイズを “-Xsfg_size_tidata_byte” サイズ以上に設定してください。
E0593003	［メッセージ］	シンボル解析情報が読み込みできません。
	［説明］	メモリ上のシンボル解析情報が存在しないか，壊れているため読み込むことができません。
	［対処方法］	CX オプションを確認後，リビルドを行ってください。
E0593004	［メッセージ］	シンボル情報ファイルが書き込みできません。
	［説明］	容量またはユーザ権限の問題が考えられます。
	［対処方法］	書き込み先の容量とユーザ権限の確認をお願いします。
E0594000	［メッセージ］	プロジェクト・ファイル (*ファイル名*) が見つかりませんでした。
	［説明］	プロジェクト・ファイルが存在しません。
	［対処方法］	ファイルが存在するか確認してください。
E0594001	［メッセージ］	プロジェクト・ファイル（*ファイル名*）の読み込みでエラーが発生しました
	［説明］	プロジェクト・ファイルの読み込み時にエラーが発生しました。 プロジェクト・ファイルが読み込み禁止である可能性があります。
E0594002	［メッセージ］	不正なフォーマットがプロジェクト・ファイル（*ファイル名*）で見つかりました。
	［説明］	プロジェクト・ファイルが，不正なフォーマットです。
	［対処方法］	プロジェクト・ファイルで不正なフォーマットを発見した場合に表示されます。該当部を修正するか，プロジェクトを作り直してください。

10.5.3 アボート・エラー

表 10.4 アボート・エラー

F0511128	[メッセージ]	Library file "*file-name*" is not found.
	[説明]	ライブラリ " *ライブラリ名* " が見つかりません。
F0512003	[メッセージ]	エラーの数が多すぎます。
F0520003	[メッセージ]	#include ファイル " *ファイル名* " は自分自身でインクルードしています。
	[説明]	#include ファイル " *ファイル名* " は自分自身でインクルードしています。修正してください。
	[メッセージ]	【CC-RX】 #include file any-string includes itself.
F0520004	[メッセージ]	メモリが足りません。
	[対処方法]	メモリが不足しています。ほかのアプリケーションを終了して，再度コンパイルし直してください。
	[メッセージ]	【CC-RX】 Out of memory.
	[対処方法]	【CC-RX】 メモリが不足しています。ほかのアプリケーションを終了して，再度コンパイルし直してください。
F0520005	[メッセージ]	ソース・ファイル " *ファイル名* " を開くことができません。
	[メッセージ]	【CC-RX】 Could not open source file "*file*".
F0520013	[メッセージ]	ファイル名がありません。
	[メッセージ]	【CC-RX】 Expected a file name.
	[説明]	【CC-RX】 ファイル名がありません。
F0520016	[メッセージ]	" *ファイル名* " は正しいソース・ファイル名ではありません。
	[メッセージ]	【CC-RX】 *ファイル名* is not a valid source file name.
	[説明]	【CC-RX】 " *ファイル名* " は正しいソース・ファイル名ではありません。
F0520035	[メッセージ]	#error 指令 : *文字列*
	[説明]	ソース・ファイル中に #error 指令がありました。
	[メッセージ]	【CC-RX】 #error directive: *文字列*
	[説明]	【CC-RX】 ソース・ファイル中に #error 指令がありました。
F0520143	[メッセージ]	プログラムはコンパイルするのに大きすぎるか複雑すぎます。
	[メッセージ]	【CC-RX】 Program too large or complicated to compile.
	[説明]	【CC-RX】 プログラムはコンパイルするのに大きすぎるか複雑すぎます。
F0520163	[メッセージ]	Could not open temporary file *xxx*.
	[説明]	テンポラリ・ファイル xxx がオープンできません。
F0520164	[メッセージ]	Name of directory for temporary files is too long (*xxx*).
	[説明]	テンポラリ・ファイル用のフォルダ名が長すぎます (*xxx*)。
F0520182	[メッセージ]	Could not open source file *xxx* (no directories in search list).
	[説明]	ソース・ファイル *xxx* を開くことができません。サーチ・リストにフォルダがありません。

F0520189	[メッセージ]	ファイル " *ファイル名* " の書き込み中にエラーが生じました。
	[メッセージ]	【CC-RX】 Error while writing *ファイル名* file.
	[説明]	【CC-RX】 ファイル " *ファイル名* " の書き込み中にエラーが生じました。
F0520190	[メッセージ]	Invalid intermediate language file.
	[説明]	不正な中間言語ファイルです。
F0520219	[メッセージ]	ファイル " *ファイル名* " の削除中にエラーが生じました。
	[メッセージ]	【CC-RX】 Error while deleting file *ファイル名* .
	[説明]	【CC-RX】 ファイル " *ファイル名* " の削除中にエラーが生じました。
F0520542	[メッセージ]	Could not create instantiation request file *名前* .
	[説明]	テンプレートを実体化するのに使用するファイル " *名前* " を作成することができませんでした。
F0520563	[メッセージ]	不正なプリプロセッサ出力ファイルです。
	[メッセージ]	【CC-RX】 Invalid preprocessor output file .
	[説明]	【CC-RX】 不正なプリプロセッサ出力ファイルです。
F0520564	[メッセージ]	プリプロセッサ出力ファイルをオープンできません。
	[メッセージ]	【CC-RX】 Cannot open preprocessor output file.
	[説明]	【CC-RX】 プリプロセッサ出力ファイルをオープンできません。
F0520583	[メッセージ]	不正なエラー出力ファイルです。
F0520584	[メッセージ]	エラー出力ファイルをオープンできません。
F0520641	[メッセージ]	*xxx* is not a valid directory.
	[説明]	*xxx* は正しいフォルダではありません。
F0520642	[メッセージ]	Cannot build temporary file name.
	[説明]	テンポラリ・ファイル名を生成できません。
F0520869	[メッセージ]	Could not set locale *xxx* to allow processing of multibyte characters.
	[説明]	マルチバイト・キャラクタを処理するロケール *xxx* がセットできません。
F0520919	[メッセージ]	Invalid output file: *xxx*
	[説明]	不正な出力ファイルです : *xxx*。
F0520920	[メッセージ]	Cannot open output file: *xxx*
	[説明]	出力ファイルがオープンできません : *xxx*。
F0520926	[メッセージ]	Cannot open definition list file: *xxx*
	[説明]	定義リスト・ファイルがオープンできません : *xxx*。
F0521083	[メッセージ]	Exported template file *xxx* is corrupted.
	[説明]	エクスポートされたテンプレート・ファイルは破損しています。
F0521151	[メッセージ]	Mangled name is too long.
	[説明]	マングル名が長すぎます。
F0521335	[メッセージ]	Cannot open predefined macro file: *xxx*
	[説明]	既定義マクロ・ファイルがオープンできません : *xxx*。

F0521336	［メッセージ］	Invalid predefined macro entry at line *行数*: *行数 2*
	［説明］	不正な定義済みマクロの entry 宣言が “ *行数* ” にあります。
F0521337	［メッセージ］	Invalid macro mode name *xxx*.
	［説明］	不正なマクロ・モード名です *xxx*。
F0521338	［メッセージ］	Incompatible redefinition of predefined macro *xxx*.
	［説明］	互換性のない定義済みマクロの再定義です。
F0523029	［メッセージ］	misra2004 ルールファイルをオープンできません。
	［説明］	-Xmisra2004=” *ファイル名* ” オプションで指定したファイルをオープンできません。
	［メッセージ］	【CC-RX】 Cannot open rule file.
	［説明］	【CC-RX】 -Xmisra2004=” *ファイル名* ” オプションで指定したファイルをオープンできません。
F0523030	［メッセージ］	ルールファイル内の記述が不正です。
	［説明］	-Xmisra2004=” *ファイル名* ” オプションで指定したファイルの内容に，不正な記述があります。
	［メッセージ］	【CC-RX】 Incorrect description *ファイル名* in rule file
	［説明］	【CC-RX】 -Xmisra2004=” *ファイル名* ” オプションで指定したファイルの内容に，不正な記述があります。
F0523031	［メッセージ］	“ *ルール番号* ” はサポートしていません。
	［説明］	サポートしていないルール番号を指定しました。
	［メッセージ］	【CC-RX】 Rule “ *ルール番号* ” is unsupported.
	［説明］	【CC-RX】 サポートしていないルール番号を指定しました。
F0523054	［メッセージ］	regID が範囲外の値です。
	［対処方法］	regID として使用可能な値を指定してください。
	［メッセージ］	【CC-RX】 regID is out of range
	［説明］	【CC-RX】 regID が範囲外の値です。
	［対処方法］	【CC-RX】 regID として使用可能な値を指定してください。
F0523055	［メッセージ］	selID が範囲外の値です
	［対処方法］	selID として使用可能な値を指定してください。
	［メッセージ］	【CC-RX】 selID is out of range
	［説明］	【CC-RX】 selID が範囲外の値です。
	［対処方法］	【CC-RX】 selID として使用可能な値を指定してください。
F0523056	［メッセージ］	第一引数が範囲外の値です
	［説明］	__set_il_rh(NUM, ADDR) の NUM として，使用不可能な値を指定しました。
	［メッセージ］	【CC-RX】 NUM is out of range.
	［説明］	【CC-RX】 __set_il_rh(NUM, ADDR) の NUM として，使用不可能な値を指定しました。
F0523061	［メッセージ］	実引数は組み込み関数の仮引数と適合しません。
F0523062	［メッセージ］	返却値の型が組み込み関数の型と合っていません。

F0523073	［メッセージ］	Illegal intrinsic function
	［説明］	サポートされていない組み込み関数を呼び出しています。
	［対処方法］	-isa オプションの指定を確認してください。
F0523300	［メッセージ］	Cannot open internal file.
	［説明］	コンパイラが内部で生成する中間ファイルをオープンすることができません。
F0523301	［メッセージ］	Cannot close internal file.
	［説明］	コンパイラが内部で生成する中間ファイルをクローズすることができません。
F0523302	［メッセージ］	Cannot write internal file.
	［説明］	中間ファイルの書き込み中にエラーが生じました。
F0530320	［メッセージ］	シンボル " *シンボル名* " が重複しています。
	［メッセージ］	【CC-RX】 Duplicate symbol " *シンボル名* "
	［説明］	【CC-RX】 シンボル " *シンボル名* " が重複しています。
F0530800	［メッセージ］	" *シンボル名* " で示すシンボルの型がファイル間で異なっています。
	［メッセージ］	【CC-RX】 Type of symbol " *シンボル名* " differs between files.
	［説明］	【CC-RX】 " *シンボル名* " で示すシンボルの型がファイル間で異なっています。
F0530808	［メッセージ］	" *変数名* " で示す変数のアライメントがファイル間で異なっています。
	［メッセージ］	【CC-RX】 Alignment of variable " *変数名* " differs between files.
	［説明］	【CC-RX】 " *変数名* " で示す変数の整列条件がファイル間で異なっています。
F0530810	［メッセージ］	" *シンボル名* " で示すシンボルの #pragma 指定がファイル間で異なっています。
	［メッセージ］	【CC-RX】 #pragma directive for symbol " *シンボル名* " differs between files.
	［説明］	【CC-RX】 " *シンボル名* " で示すシンボルの #pragma 指定がファイル間で異なっています。
F0531003	［メッセージ］	*オプション*オプションで指定された関数*関数*が見つかりません。
F0533015	［メッセージ］	シンボル数が限界値を越えました。
	［説明］	コンパイラが生成するシンボルの数が限界値を越えました。
F0533021	［メッセージ］	メモリが足りません。
	［対処方法］	ほかのアプリケーションを終了して，再度コンパイルし直してください。
	［メッセージ］	【CC-RX】 Out of memory.
	［対処方法］	【CC-RX】 メモリが不足しています。ほかのアプリケーションを終了して，再度コンパイルし直してください。
F0533300	［メッセージ］	Cannot open an intermediate file.
	［説明］	コンパイラが内部で生成する中間ファイルをオープンすることができません。
F0533301	［メッセージ］	中間ファイルをクローズできません。
	［説明］	コンパイラが内部で生成する中間ファイルをクローズすることができません。
	［メッセージ］	【CC-RX】 Cannot close an intermediate file.
	［説明］	【CC-RX】 コンパイラが内部で生成する中間ファイルをクローズすることができません。

F0533302	［メッセージ］	中間ファイルの読み込み中にエラーが生じました。
	［メッセージ］	【CC-RX】 Cannot read an intermediate file.
	［説明］	【CC-RX】 中間ファイルの読み込み中にエラーが生じました。
F0533303	［メッセージ］	中間ファイルの書き込み中にエラーが生じました。
	［メッセージ］	【CC-RX】 Cannot write to an intermediate file.
	［説明］	【CC-RX】 中間ファイルの書き込み中にエラーが生じました。
F0533306	［メッセージ］	コンパイル処理を中断しました。
	［説明］	コンパイル処理中に Cntl + C コマンドによる割り込みを検出しました。
	［メッセージ］	【CC-RX】 Compilation was interrupted.
	［説明］	【CC-RX】 コンパイル処理中に Cntl + C コマンドによる割り込みを検出しました。
F0533330	［メッセージ］	中間ファイルをオープンできません。
	［説明］	コンパイラが内部で生成する中間ファイルをオープンすることができません。
F0540027	［メッセージ］	ファイル ” *ファイル名* ” が読み込みできません。
	［メッセージ］	【CC-RX】 Cannot read file " *ファイル名* ".
	［説明］	【CC-RX】 ファイル ” *ファイル名* ” が読み込みできません。
F0540204	［メッセージ］	【CC-RH】 関数内で使用するスタックのサイズが大きすぎます。
	［説明］	【CC-RH】 関数内で使用するスタックのサイズが 2G バイトを越えています。
	［メッセージ］	【CC-RX】 Illegal stack access.
	［説明］	【CC-RX】 関数内で使用するスタックのサイズが 2G バイトを越えています。
	［メッセージ］	【CC-RH】 関数内で使用するスタックのサイズが大きすぎます。
	［説明］	【CC-RH】 関数内で使用するスタックのサイズが 64K バイトを越えています。
F0540300	［メッセージ］	中間ファイルをオープンできません。
	［説明］	コンパイラが内部で生成する中間ファイルをオープンすることができません。
	［メッセージ］	【CC-RX】 Cannot open an intermediate file.
	［説明］	【CC-RX】 コンパイラが内部で生成する中間ファイルをオープンすることができません。
F0540301	［メッセージ］	中間ファイルをクローズできません。
	［説明］	コンパイラが内部で生成する中間ファイルをクローズすることができません。
	［メッセージ］	【CC-RX】 Cannot close an intermediate file.
	［説明］	【CC-RX】 コンパイラが内部で生成する中間ファイルをクローズすることができません。
F0540302	［メッセージ］	中間ファイルの読み込み中にエラーが生じました。
	［メッセージ］	【CC-RX】 Cannot read an intermediate file.
	［説明］	【CC-RX】 中間ファイルの読み込み中にエラーが生じました。
F0540303	［メッセージ］	中間ファイルの書き込み中にエラーが生じました。
	［メッセージ］	【CC-RX】 Cannot write to an intermediate file.
	［説明］	【CC-RX】 中間ファイルの書き込み中にエラーが生じました。

F0540400	［メッセージ］	#pragma " *識別子名* " に異なるパラメータが設定されています。
	［メッセージ］	【CC-RX】 Different parameters are set for the same #pramga " *識別子名* ".
	［説明］	【CC-RX】 #pragma " *識別子名* " に異なるパラメータが設定されています。
F0542001	［メッセージ］	-Xr *数値* オプションで指定された " *シンボル* " を、レジスタに割り付けることができません。
F0542002	［メッセージ］	ビット・フィールドの初期値にアドレス定数を記述できません。
F0544302	［メッセージ］	Cannot read an intermediate file.
	［説明］	中間ファイルの読み込み中にエラーが生じました。
F0544802	［メッセージ］	The value of the parameter for the in-line function is outside the defined range.
	［説明］	組み込み関数の引数に指定した値が，関数で定義された範囲を越えています。
F0550503	［メッセージ］	ファイル *file* をオープンできません。
	［対処方法］	ファイルを確認してください。
F0550504	［メッセージ］	セクション定義疑似命令においてセクションの種類の指定に誤りがあります。
	［対処方法］	セクションの種類の指定を確認してください。
F0550505	［メッセージ］	メモリが足りません。
	［対処方法］	空きメモリを確認してください。
F0550506	［メッセージ］	内部データ領域（*string*）の確保に失敗しました。
	［対処方法］	空きメモリを確認してください。
F0550507	［メッセージ］	式の処理において作業領域が足りなくなりました。(*string*)
	［説明］	式の処理において作業領域が足りなくなりました。単純な式に変更してください。
	［対処方法］	式を確認してください。
F0550508	［メッセージ］	定義されていない識別子 *identifier* が参照されています。
	［対処方法］	識別子を確認してください。
F0550509	［メッセージ］	予期しない疑似命令 *string* が見つかりました。
	［対処方法］	疑似命令を確認してください。
F0550510	［メッセージ］	*string* 疑似命令に対応する疑似命令が存在しません。
	［対処方法］	疑似命令を確認してください。
F0550511	［メッセージ］	条件アセンブル制御命令において対応する疑似命令 *string* が存在しません。
	［対処方法］	条件アセンブル制御命令を確認してください。
F0550512	［メッセージ］	条件アセンブル制御命令が 4294967294 回以上ネストして用いられています。
	［対処方法］	ネストを確認してください。
F0550513	［メッセージ］	*string* 疑似命令に対応する .endm 疑似命令が存在しません。
	［対処方法］	疑似命令を確認してください。
F0550514	［メッセージ］	実引数が 4294967294 個以上用いられています。
	［対処方法］	実引数を確認してください。
F0550515	［メッセージ］	*string* 疑似命令が .rept/.irp 疑似命令に囲まれていません。
	［対処方法］	疑似命令を確認してください。

F0550516	［メッセージ］	.local 疑似命令により自動生成されたシンボルが限界数（4294967294）を越えました。
	［対処方法］	疑似命令を確認してください。
F0550517	［メッセージ］	*string* が 4294967294 回以上ネストして用いられています。
	［対処方法］	ネストを確認してください。
F0550518	［メッセージ］	マクロ本体内で現在定義中のマクロの呼び出し *macro_call* が行われました。
	［対処方法］	マクロ定義を確認してください。
F0550519	［メッセージ］	マクロ呼び出しの引数指定が不正です。
	［対処方法］	引数指定を確認してください。
F0550520	［メッセージ］	'$' のあとにシンボル以外の識別名、または未定義シンボル名が指定されています。
	［対処方法］	記述を確認してください。
F0550521	［メッセージ］	致命的なエラーの数が 30 に達しました。アセンブルを中止します。
	［対処方法］	エラーを修正してください。
F0550523	［メッセージ］	オプション、または疑似命令でターゲット・デバイス指定が重複しています。
	［対処方法］	ターゲット・デバイス指定を確認してください。
F0550524	［メッセージ］	指定されたターゲット・デバイスに相当するデバイス・ファイルがないか、デバイス指定が間違っている、またはデバイス指定がありません。
	［対処方法］	デバイス・ファイルを確認してください。
F0550525	［メッセージ］	指定されたデバイス・ファイルが 対応マイクロコントローラではありません。
	［対処方法］	デバイス・ファイルを確認してください。
F0550526	［メッセージ］	指定されたデバイス・ファイルのバージョンが不正です。バージョン *version* は指定できません。
	［対処方法］	デバイス・ファイルを確認してください。
F0550528	［メッセージ］	tidata.byte セクション、tibss.byte セクションのサイズの合計が 128 バイトを越えています。
	［対処方法］	tidata.byte セクション，tibss.byte セクションのサイズを確認してください。
F0550529	［メッセージ］	*セクション名 1* セクション、*セクション名 2* セクションのサイズの合計が 256 バイトを越えています。
	［対処方法］	*セクション名 1* セクション，*セクション名 2* セクションのサイズを確認してください。
F0550530	［メッセージ］	tidata.byte セクション、tibss.byte セクション、tidata.word セクション、tibss.word セクション、tidata セクション、tibss セクションのサイズの合計が 256 バイトを越えています。
	［対処方法］	tidata.byte セクション，tibss.byte セクション，tidata.word セクション，tibss.word セクション，tidata セクション，tibss セクションのサイズを確認してください。
F0550531	［メッセージ］	1 ファイルに記述できるシンボル数を越えました。記述できるシンボル数の限界は、アセンブラが内部で登録するものを含め、4294967294 です。
F0550532	［メッセージ］	リンク可能なオブジェクト・ファイルを生成する段階で、ファイル・システムに依存するエラーが発生しました。
	［対処方法］	ファイル・システムを確認してください。

F0550533	［メッセージ］	関数 *function* のサイズがサポート外です。スマートコレクションは利用できません。
	［対処方法］	関数 *function* のサイズを確認してください。
F0550534	［メッセージ］	1 ファイル中の命令の数が多すぎます。
	［説明］	1 ファイル中の命令の数が限界を越えています。限界値は 10,000,000 です。
	［対処方法］	命令の数を確認してください。
F0550535	［メッセージ］	'*function1*' と '*function2*' が同じセクションに定義されています。
	［対処方法］	$smart_correct 制御命令で指定した修正後関数は，修正前関数とは別のセクションに記述してください。
F0550536	［メッセージ］	このデバイスでは、*Option* オプションをサポートしていません。
	［対処方法］	ターゲット・デバイスを確認してください。
F0550537	［メッセージ］	セクション (*section*) のアドレスが使用可能なアドレス範囲を超えました。
	［説明］	絶対アドレス指定セクションの配置アドレスが 0xffffffff を越えています。
	［対処方法］	.org によるセクションの絶対アドレス指定は，セクションの終端命令の配置アドレスが 0xffffffff までになるように行ってください。
F0550538	［メッセージ］	セクション (*section1*) と (*section2*) のアドレスが重複しました。
	［説明］	絶対アドレス指定セクションの配置アドレス範囲が，ほかのセクションの配置アドレス範囲と重複しています。
	［対処方法］	.org で指定しているアドレスを確認してください。
F0550539	［メッセージ］	リロケーションを出力可能なシンボルが限界数 (16777215) を越えました。
	［説明］	16777216 個以上のシンボルが登録されており，かつ参照されています。
	［対処方法］	シンボル数を確認してください。
F0550540	［メッセージ］	ファイル *file* の読み込みができません。
	［説明］	ファイルが不正か，ファイルのサイズが読み込みできる限界を越えているかもしれません。
	［対処方法］	ファイルを確認してください。
F0550703	［メッセージ］	エラーの数が多すぎます。
F0551601	［メッセージ］	"*source*" で指定されたデバイス情報に誤りがあります。
F0551602	［メッセージ］	シンボル名が長すぎます。
F0551604	［メッセージ］	RL78-S1 コア指定時は -mirror_source=1 の指定はできません。
F0551605	［メッセージ］	-dev 指定時は -mirror_region の指定はできません。
F0551606	［メッセージ］	-mirror_source=common 指定時は -mirror_region の指定はできません。
F0551607	［メッセージ］	ミラー領域として入力された値に誤りがあります。
F0551608	［メッセージ］	アドレスを指定してください。
F0551609	［メッセージ］	インクルード・ファイルの入れ子回数が上限を超えました。
	［説明］	インクルードの階層が深すぎるか，自分自身を再帰的にインクルードしている可能性があります。
	［対処方法］	インクルード・ファイルを見直してください。

F0551610	[メッセージ]	マクロ呼び出しの入れ子回数が上限を超えました。
	[説明]	マクロ呼び出しの階層が深すぎるか，自分自身を再帰的に呼び出している可能性があります。
	[対処方法]	マクロ定義を見直してください。
F0553000	[メッセージ]	Can't create file '*filename*'.
	[説明]	'*filename*' ファイルが生成できません。
	[対処方法]	ディレクトリ容量を確認してください。
F0553001	[メッセージ]	Can't open file '*filename*'.
	[説明]	'*filename*' ファイルがオープンできません。
	[対処方法]	ファイル名を確認してください。
F0553002	[メッセージ]	Can't write file '*filename*'.
	[説明]	'*filename*' ファイルに書き込むことができません。
	[対処方法]	ファイルのパーミッションを確認してください。
F0553003	[メッセージ]	Can't read file '*filename*'.
	[説明]	ファイルを読み込むことができません。
	[対処方法]	ファイルのパーミッションを確認してください。
F0553004	[メッセージ]	Can't create Temporary file.
	[説明]	テンポラリ・ファイルが生成できません。
	[対処方法]	カレントディレクトリ以外にテンポラリ・ファイルを作成するように，環境変数 'TMP_RX' にディレクトリを指定してください。
F0553005	[メッセージ]	Can't open Temporary file.
	[説明]	テンポラリ・ファイルがオープンできません。
	[対処方法]	'TMP_RX' で指定したディレクトリを確認してください。
F0553006	[メッセージ]	Can't read Temporary file.
	[説明]	テンポラリ・ファイルを読み込むことができません。
	[対処方法]	'TMP_RX' で指定したディレクトリを確認してください。
F0553007	[メッセージ]	Can't write Temporary file.
	[説明]	テンポラリ・ファイルに書き込むことができません。
	[対処方法]	'TMP_RX' で指定したディレクトリを確認してください。
F0553008	[メッセージ]	Illegal file name '*filename*'.
	[説明]	ファイル名が不正です。
	[対処方法]	ファイル名の記述規則に従ったファイル名を指定してください。
F0553016	[メッセージ]	Lacking cpu specification.
	[説明]	CPU の指定がされていません。
	[対処方法]	cpu オプション，または環境変数 CPU_RX で CPU を指定してください。

F0553100	［メッセージ］	Command line is too long.
	［説明］	コマンド行の文字数が多すぎます。
	［対処方法］	コマンドを入力し直してください。
F0553101	［メッセージ］	Invalid option '*xx*' is used.
	［説明］	無効なコマンドオプション *xx* を使用しています。
	［対処方法］	指定したオプションは存在しません。コマンドを入力し直してください。
F0553102	［メッセージ］	Ignore option '*xx*'.
	［説明］	無効なオプションが指定されています。
F0553103	［メッセージ］	Option '*xx*' is not appropriate.
	［説明］	コマンドオプション *xx* の記述が正しくありません。
	［対処方法］	コマンドオプションを指定し直してください。
F0553104	［メッセージ］	No input files specified.
	［説明］	入力ファイルの指定がありません。
	［対処方法］	入力ファイルを指定してください。
F0553105	［メッセージ］	Source files number exceed 80.
	［説明］	ファイルの数が 80 を越えています。
	［対処方法］	複数回にわけてアセンブルを実行してください。
F0553106	［メッセージ］	Lacking cpu specification.
	［説明］	CPU の指定がされていません。
	［対処方法］	cpu オプション，または環境変数 CPU_RX で CPU を指定してください。
F0553110	［メッセージ］	Multiple register base/fint_register.
	［説明］	base と fint_register オプションで指定レジスタが重複しています。
F0553111	［メッセージ］	Multiple register base/pid.
	［説明］	base と pid オプションで指定レジスタが重複しています。
F0553112	［メッセージ］	Multiple register base/nouse_pid_register.
	［説明］	base と nouse_pid_register オプションで指定レジスタが重複しています。
F0553113	［メッセージ］	Neither isa nor cpu is specified
	［説明］	-isa オプション，-cpu オプションのいずれの指定もありません。
F0553114	［メッセージ］	Both '-isa' option and '-cpu' option are specified
	［説明］	-isa オプションと -cpu オプションを同時に指定しています。
F0553115	［メッセージ］	The '-cpu' option and the '-fpu' option are inconsistent
	［説明］	-cpu（CPU_RX）の指定と -fpu の指定が矛盾しています。
F0553200	［メッセージ］	Error occurred in executing '*xxx*'.
	［説明］	*xxx* の実行でエラーが発生しました。
	［対処方法］	再度 asrx を実行し直してください。

F0553201	[メッセージ]	Not enough memory.
	[説明]	メモリが足りません。
	[対処方法]	ファイルを分割して実行し直してください。または，メモリを増設してください。
F0553202	[メッセージ]	Can't find work dir.
	[説明]	ワークディレクトリが見つかりません。
	[対処方法]	環境変数 TMP_RX が正しく設定されているかを確認してください。
F0560001	[メッセージ]	入力ファイル “*file*” をオープンできません。
F0560002	[メッセージ]	出力ファイル “*file*” をオープンできません。
F0560003	[メッセージ]	出力ファイル “*file*” をクリエイトできません。
F0560004	[メッセージ]	ディレクティブファイル “*file*” をオープンできません。
F0560005	[メッセージ]	ディレクティブファイル “*file*” のサイズの取得に失敗しました。
F0560006	[メッセージ]	出力ファイル “*file*” のサイズを *number* バイトに変更できません。
F0560007	[メッセージ]	出力ファイル “*file*” をシークできません。
F0560008	[メッセージ]	出力ファイル “*file*” の書き込みができません。
F0560009	[メッセージ]	デバイス・ファイル “*string*” が見つかりません。
F0560010	[メッセージ]	デバイス・ファイル “*string*” が不正です。
F0560011	[メッセージ]	デバイス・ファイル “*string*” をオープンできません。
F0560012	[メッセージ]	デバイス・ファイル “*string*” の読み込みができません。
F0560013	[メッセージ]	オブジェクト・ファイルが不正です。(エラー番号 : *number*)
F0560101	[メッセージ]	ELF 形式の版が本リンカで扱うことのできる版ではありません。
F0560102	[メッセージ]	入力ファイルのタイプが本リンカで扱うことのできるタイプではありません。
F0560103	[メッセージ]	入力ファイルのタイプが本リンカで扱うことのできるクラスではありません。
F0560104	[メッセージ]	入力ファイルのバイト・オーダが本リンカで扱うことのできるバイト・オーダではありません。
F0560105	[メッセージ]	入力ファイルとして扱うことのできるオブジェクト・ファイルのファイル・タイプは、リロケータブル・ファイルまたは共有ライブラリ・ファイルのみです。
F0560106	[メッセージ]	指定されたファイル “*file*” は不正なファイル形式を持っています。
F0560107	[メッセージ]	ターゲットのマイクロコントローラ共通のマジック・ナンバ “*string*” でリンクできないファイルが入力されました。
F0560108	[メッセージ]	“*file*” は実行形式ファイルではありません。ブート・ファイルとして入力することはできません。
F0560109	[メッセージ]	“*file*” は ROM 化済みの実行形式ファイルです。ブート・ファイルとして入力することはできません。
F0560110	[メッセージ]	同じファイルが入力ファイルとして複数回指定されています。
F0560112	[メッセージ]	*string1* コアの共通オブジェクト・ファイルと *string2* コアのオブジェクト・ファイルをリンクすることはできません。ファイル “*file*” は *string1* コアの共通オブジェクト・ファイルです。 -Xcommon オプションで指定されたマジック・ナンバの混合チェックでエラーが発生しました。

F0560113	[メッセージ]	マスク・レジスタを使用したオブジェクト・ファイルと使用していないオブジェクト・ファイルをリンクすることはできません。ファイル “*file*” はマスク・レジスタを使用したオブジェクト・ファイルです。
F0560114	[メッセージ]	BPC 値の異なるファイルが入力されています。
F0560115	[メッセージ]	FPU を使用したオブジェクト・ファイルを指定されたターゲット・デバイスでリンクすることはできません。
F0560201	[メッセージ]	シンボル名文字列の取得に失敗しました。
F0560202	[メッセージ]	*number* 番目のシンボル名文字列の取得に失敗しました。
F0560203	[メッセージ]	シンボル “*symbol*” は、不正なバインディング・クラス (*number*) を持っています。
F0560204	[メッセージ]	WEAK のバインディング・タイプを持つシンボル “*symbol*” はサポートしていません。
F0560208	[メッセージ]	シンボル “*symbol*” は、多重に定義されています。
F0560209	[メッセージ]	*file1* 内で定義されたシンボル “*symbol*” と *file2* 内で定義された同じ名前のシンボルのセクション配置に矛盾が生じました。
F0560210	[メッセージ]	-Xentry_address オプションで指定されたシンボル “*symbol*” が見つかりません。
F0560213	[メッセージ]	$EXT_FUNC 制御命令で指定されたシンボル名 *symbol1* と ID 値 *ID1* と他のファイルで定義済みのシンボル名 *symbol2* と ID 値 *ID2* に矛盾があります。
F0560215	[メッセージ]	シンボル数が限界値を越えました。
F0560216	[メッセージ]	シンボルの多重定義をすべて表示します。 シンボル　定義ファイル 1　定義ファイル 2 *symbol*　*file1*　*file2*
F0560217	[メッセージ]	ファイル file 内で参照されているシンボル “*symbol*” が定義されていません。
F0560218	[メッセージ]	シンボル・テーブルの領域が足りなくなりました。
F0560219	[メッセージ]	$EXT_ENT_SIZE で指定されたサイズと “*symbol*” のサイズが異なります。
F0560220	[メッセージ]	GP-symbol"*symbo2*" のベース・シンボルとして指定された TP-symbol"*symbol*" が見つかりません。
F0560301	[メッセージ]	セクション名のストリング・テーブル・セクションの取得に失敗しました。
F0560302	[メッセージ]	シンボル名のストリング・テーブル・セクションの取得に失敗しました。
F0560303	[メッセージ]	セクション・ヘッダの取得に失敗しました。
F0560304	[メッセージ]	セクション名の取得に失敗しました。
F0560306	[メッセージ]	セクション “*section*” は、不正なセクション・タイプ (*number*) を持っています。
F0560307	[メッセージ]	セクション “*section*” のロウ・データの取得に失敗しました。
F0560309	[メッセージ]	割り込みハンドラセクション “*secton*” はすでに定義されています。本リンカでは割り込みハンドラセクションを自動生成するので、リンク・ディレクティブへの記述は不要です。
F0560310	[メッセージ]	特殊セクション “*secton*” は、すでに定義されています。
F0560311	[メッセージ]	セクション “*secton*” のサイズが不正です。
F0560404	[メッセージ]	分岐系のリロケーション・エントリ (シンボル :*symbol*, ファイル :file, セクション :*section*, オフセット :offset, タイプ :*relocation type*) によってリロケートされた値 (*value*) が奇数になっています。

F0560405	[メッセージ]	GP 相対リロケーションのための GP シンボルが存在しません (LOCAL) (ファイル : *file2*, セグメント :*segment*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。ローカル・シンボルは "*section1*"(ファイル :*file1*) に存在します。
F0560407	[メッセージ]	GP 相対リロケーションのための GP シンボルが存在しません (GLOBAL) 。 (ファイル : *file2*, セグメント :*segment*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。シンボルは "*section1*"(ファイル :*file1*) に存在します。
F0560409	[メッセージ]	ブート側実行ファイルのセクション "*section*" に GP 相対リロケーションのための GP シンボルが存在しません。 (ファイル : *file2*, セグメント :*segment*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。シンボルは "*section1*"(ファイル :*file1*) に存在します。
F0560411	[メッセージ]	分岐系のリロケーション・エントリ（ファイル :*file*, セクション :*section*, オフセット :*offset*, リロケーション・タイプ :*relocation type*）によってリロケートされた値 (*value*) が奇数になっています。
F0560414	[メッセージ]	EP シンボルが生成されていません。
F0560415	[メッセージ]	出力セクション *section2* から、出力セクション *section1* に配置されたシンボル "*symbol*" への分岐命令が、分岐可能範囲を越えています。分岐命令は (値 :*value*, ファイル :*file* 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。
F0560417	[メッセージ]	出力セクション *section2* から、出力セクション *section1* に配置されたローカル・シンボルへの分岐命令が、分岐可能範囲を越えています。分岐命令は (値 :*value*, ファイル :file 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。
F0560419	[メッセージ]	出力セクション *section1* がオーバフローしました。または、シンボル "*symbol*" の配置されたセクションとラベルの参照方法がファイル "*file*" において不正です。不正な参照は、(値 :*value*, 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。シンボル "*symbol*" はセクション "*section1*" に配置されました (ファイル :*file1*)。
F0560421	[メッセージ]	出力セクション *section1* がオーバフローしました。または、ローカル・シンボルの配置されたセクションとラベルの参照方法がファイル "*file2*" において不正です。不正な参照は、(値 :*value*, 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。ローカル・シンボルはセクション "*section1*" に配置されました (ファイル :*file1*)。
F0560423	[メッセージ]	ブート - フラッシュ再リンク機能において、ブート領域側作成時にフラッシュ領域側のシンボル "*symbol*" を string による参照はできません。
F0560424	[メッセージ]	ブートファイル "*file*" にはシンボル情報がありません。
F0560502	[メッセージ]	ライブラリ・シンボル・テーブル・エントリで指定されたオフセット (offset) の位置にライブラリ・メンバが見つかりません。ライブラリ・シンボル・テーブルが破壊されているおそれがあります。
F0560503	[メッセージ]	共有ライブラリはサポートしていません。
F0560627	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域がターゲット・マシンにおける可能なメモリ空間の範囲を越えています。
F0560629	[メッセージ]	セグメント "*segment*" のセグメント・ディレクティブには *string* が必要とされています。
F0560630	[メッセージ]	セクション・ディレクティブには *string* が必要とされています。
F0560631	[メッセージ]	シンボル "*symbol*" のシンボル・ディレクティブには *string* が必要とされています。
F0560633	[メッセージ]	セグメント "*segment1*" の先頭アドレス (*number1*) が、アドレス (*number2*) の前までに割り付けられているセグメント "*segment2*" の領域にオーバラップしています。

F0560634	[メッセージ]	セクション “*section1*” の先頭アドレス (*number1*) が、アドレス (*number2*) の前までに割り付けられているセクション “*section2*” の領域にオーバラップしています。
F0560635	[メッセージ]	セクション “*section*” の先頭アドレス (*number1*) が、属するセグメント “*segment*” の先頭アドレス (*number2*) よりも手前に割り付けられています。
F0560636	[メッセージ]	セグメント “*segment*” のメモリ・サイズ (*number1*) が、明示的に指定された最大メモリ・サイズまたはデフォルトの最大メモリ・サイズ (*number2*) を越えています。
F0560649	[メッセージ]	セクション “*section*” の終端アドレスが最大メモリアドレス (*number*) をオーバフローしました。
F0560650	[メッセージ]	セグメント “*segment*” の終端アドレスが最大メモリアドレス (*number*) をオーバフローしました。
F0560652	[メッセージ]	セグメント “*segment*” の割り付けが指示された領域 (*number1-number2*) がターゲット・マシンにおける可能な内蔵メモリ空間の範囲 (*number3-number4*) を越えています。
F0560657	[メッセージ]	セクション “section” は、ディレクティブ・ファイルで指定されている必要があります。
F0560658	[メッセージ]	ディレクティブ・ファイル中のセクション “*section*” の記述が不正です。
F0560661	[メッセージ]	メモリ領域の確保に失敗しました。
F0560662	[メッセージ]	CALLT ベース・ポインタ (CTBP) が設定されていません。-Xpro_epi_runtime=off または -Ospeed 指定により callt 命令の出力を抑制するか、CTBP を設定してください。
F0560701	[メッセージ]	*string2* オプションが指定された場合、*string1* オプションを指定することはできません。
F0560705	[メッセージ]	ターゲット・デバイスを指定してください。 このメッセージは、リンク可能なオブジェクト・ファイル生成時に、共通マジック・ナンバが埋め込まれたファイルのみをリンクして、実行可能なオブジェクト・ファイルを生成しようとした場合に出力されます。
F0560707	[メッセージ]	“$ext_func” 指定のあるファイルに、“ -Xrelinkable_object” を指定することはできません。
F0560708	[メッセージ]	“*string1*” オプションは、“*string2*” オプションを必要とします。
F0560712	[メッセージ]	“*string*” のオプションに指定した値がオーバフローしました。
F0561001	[メッセージ]	入力ファイル “*file*” は不正であるため入力できません。
F0561002	[メッセージ]	リロケータブル・オブジェクト “*file*” が入力ファイルとして指定されています。
F0561003	[メッセージ]	不正な入力ファイルが指定されました。ファイル “*file*” はライブラリファイルです。
F0561004	[メッセージ]	メモリの確保に失敗しました。
F0561005	[メッセージ]	“*symbol*” のアドレスは、すべての入力ファイルで同じにしてください。
F0561006	[メッセージ]	指定したシンボル “*symbol*” がオブジェクト・ファイル “*file*” 内に見つかりませんでした。
F0561007	[メッセージ]	rompsec セクションを作成する際にメモリの上限を越えました。 -Xromize_check_off=rom_less オプションを指定することにより、本メッセージを消去することが可能です。
F0561008	[メッセージ]	“*section1*” セクションと “*section2*” セクションが重なっています。
F0561009	[メッセージ]	start_label ラベルは、4 バイト境界のアドレスにあるようにしてください。
F0561010	[メッセージ]	指定したセクションがオブジェクト・ファイル内に見つかりませんでした。

F0561011	[メッセージ]	内蔵命令 RAM に配置される “*section*” セクションがワードアライメントされていません。
F0561012	[メッセージ]	セクションがみつかりません。
F0561013	[メッセージ]	file“*file*” 内の -Xrompsec_data オプションで指定された “*section*" は、-Xrompsec_data オプションでは指定できないセクション属性です。
F0561014	[メッセージ]	file“*file*” 内の -Xrompsec_text オプションで指定された “*section*” は、-Xrompsec_text オプションでは指定できないセクション属性です。
F0561015	[メッセージ]	-Xrompsec_data オプションで指定された “*section*” セクションが見つかりませんでした。
F0561016	[メッセージ]	-Xrompsec_text オプションで指定された “*section*” セクションが見つかりませんでした。
F0561019	[メッセージ]	プロセッサ・タイプはすべてのファイルにおいて同一でなければなりません。
F0562001	[メッセージ]	-Xhex_fill オプションで指定された領域（*address1-address2*）にセクションがありません。
F0562002	[メッセージ]	“*section*” セクションが見つかりません。
F0562003	[メッセージ]	“*file*” は、ELF 形式のオブジェクト・ファイルではありません。
F0562004	[メッセージ]	“*file*” は、ライブラリ・ファイルです。ライブラリ・ファイルを指定することはできません。
F0562005	[メッセージ]	オブジェクトファイル (*string*) が不正です。
F0562006	[メッセージ]	ファイル名 “*name*” の長さが限界を越えているため、モトローラ S タイプ・ヘキサ・フォーマットの S0 レコードに登録することができません。ファイル名は 254 文字以下にしてください。
F0562007	[メッセージ]	メモリ情報がないため、ROM データを作成できません。
F0562008	[メッセージ]	指定されたセクション “*section*” が見つかりません。
F0562009	[メッセージ]	メモリが足りません。
F0562014	[メッセージ]	-Xhex_fill 指定時はエラー “*section*” セクションは、-Xhex_fill オプションで指定した領域の下限を越えています。 -Xhex_fill 指定時されてなければワーニング “*section*” セクションは、内蔵 ROM 領域の下限を越えています。
F0562028	[メッセージ]	CRC *xxx* オプションのパラメータ *xxx* 指定にエラーがあります。
F0563000	[メッセージ]	No input file
	[説明]	入力ファイルがありません。
F0563001	[メッセージ]	No module in library
	[説明]	ライブラリ内のモジュール数が 0 になりました。
F0563002	[メッセージ]	Option " *オプション1*" is ineffective without option " *オプション2*"
	[説明]	" *オプション1*" は " *オプション2*" が必要です。
F0563004	[メッセージ]	Unsupported inter-module optimization information type " *タイプ*" in " *ファイル* "
	[説明]	ファイル内にサポートしていないモジュール間最適化情報 " *タイプ*" がありました。
	[対処方法]	コンパイラ，アセンブラのバージョンが正しいか確認してください。

F0563010	[メッセージ]	No mirror region information
	[説明]	ミラー領域の配置アドレス情報が指定されていません。
	[対処方法]	-far_rom，-mirror_region，または -dev オプションが正しいか確認してください。
F0563020	[メッセージ]	No cpu information in input files
	[説明]	CPU 種別を入力ファイルから識別できません。
	[対処方法]	入力ファイルを確認してください。
F0563100	[メッセージ]	Section address overflow out of range : " *セクション*"
	[説明]	" *セクション*" のアドレスが使用可能な上限の領域を越えました。
	[対処方法]	start オプションのアドレス指定を変更してください。 なお，アドレス空間の詳細についてはデバイスのユーザーズ・マニュアルを参照してください。
F0563102	[メッセージ]	Section contents overlap in absolute section " *セクション*"
	[説明]	絶対アドレス・セクションのセクション内データ・アドレスが重複しています。
	[対処方法]	ソース・プログラムを修正してください。
F0563110	[メッセージ]	Illegal cpu type " *マイコン種別*" in " *ファイル*"
	[説明]	異なるマイコン種別のファイルを入力しました。
F0563111	[メッセージ]	Illegal encode type " *エンディアン種別*" in " *ファイル*"
	[説明]	異なるエンディアン種別のファイルを入力しました。
F0563112	[メッセージ]	Invalid relocation type in " *ファイル*"
	[説明]	" *ファイル*" 内にサポートしていないリロケーション・タイプがありました。
	[対処方法]	コンパイラ，アセンブラのバージョンが正しいか確認してください。
F0563113	[メッセージ]	Illegal mode type " *モード*" in " *ファイル*"
	[説明]	混在できない " *モード*" のファイルを入力しました。
	[対処方法]	コンパイラ，アセンブラのオプション，およびデバイス・ファイルが正しいか確認してください。
F0563114	[メッセージ]	Illegal cpu type "*CPU 種別*" in device file " *ファイル*"
	[説明]	"*CPU 種別*" が異なります。
	[対処方法]	デバイス・ファイルが正しいか確認してください。
F0563120	[メッセージ]	Illegal size of the absolute code section : " *セクション*" in " *ファイル*"
	[説明]	" *ファイル*" に存在する絶対アドレス・プログラム・セクション " *セクション*" のサイズが不正です。
F0563121	[メッセージ]	Illegal type of the section : " *セクション*" in " *ファイル*"
	[説明]	“ *セクション*” の種類が異なります。
	[対処方法]	ソース・ファイルのセクション指定を確認してください。
F0563122	[メッセージ]	Illegal attribute of the section : " *セクション*" in " *ファイル*"
	[説明]	“ *セクション*” の種類が異なります。
	[対処方法]	ソース・ファイルのセクション指定を確認してください。

F0563123	[メッセージ]	Gap is within the limits of the section : " *セクション* "
	[説明]	" *セクション* " を配置することができません。
	[対処方法]	ソース・ファイルを確認してください。
F0563124	[メッセージ]	Illegal alignment of the section : " *セクション* "
	[説明]	" *セクション* " のアライメントにより配置することができません。
	[対処方法]	ソース・ファイルを確認してください。
F0563125	[メッセージ]	Illegal kind of the section : " *セクション* " in " *ファイル* "
	[説明]	" *セクション* " の種類が異なります。
	[対処方法]	ソース・ファイルのセクション指定を確認してください。
F0563130	[メッセージ]	Range " *領域* " in " *ファイル* " conflicts with that in another file
	[説明]	" *領域* " のメモリ範囲が異なるファイルを入力しました。
	[対処方法]	コンパイラ，アセンブラのオプションが正しいか確認してください。
F0563140	[メッセージ]	No " *領域* " area information in input device file
	[説明]	" *領域* " の情報がデバイス・ファイルにありません。
	[対処方法]	デバイス・ファイルが正しいか確認してください。
F0563200	[メッセージ]	Too many sections
	[説明]	セクション数が翻訳限界を越えました。複数ファイル出力を指定すると解決できる可能性があります。
F0563201	[メッセージ]	Too many symbols
	[説明]	シンボル数が翻訳限界を越えました。複数ファイル出力を指定すると解決できる可能性があります。
F0563202	[メッセージ]	Too many modules
	[説明]	モジュール数が翻訳限界を越えました。
	[対処方法]	ライブラリを分けて作成してください。
F0563203	[メッセージ]	Reserved module name "rlink_generates"
	[説明]	rlink_generates_** （** は，01 ～ 99 までの数値）は，最適化リンカで使用する予約名称です。.obj/.rel ファイル名，およびライブラリ内モジュール名として使用しています。
	[対処方法]	ファイル名，およびライブラリ内モジュール名で使用している場合は，変更してください。
F0563204	[メッセージ]	Reserved section name "$sss_fetch"
	[説明]	sss_fetch** （sss は任意の文字列，** は 01 ～ 99 までの数値）は，最適化リンカで使用する予約名称です。
	[対処方法]	シンボル名，またはセクション名を変更してください。
F0563300	[メッセージ]	Cannot open file : " *ファイル* "
	[説明]	" *ファイル* " をオープンできません。
	[対処方法]	ファイル名，およびアクセス権が正しいか，確認してください。
F0563301	[メッセージ]	Cannot close file : " *ファイル* "
	[説明]	" *ファイル* " をクローズできません。ディスク容量に空きがない可能性があります。

F0563302	[メッセージ]	Cannot write file : " *ファイル* "
	[説明]	" *ファイル* " に書き込めません。ディスク容量に空きがない可能性があります。
F0563303	[メッセージ]	Cannot read file : " *ファイル* "
	[説明]	" *ファイル* " を読めません。空ファイルを入力したか，ディスク容量に空きがない可能性があります。
F0563310	[メッセージ]	Cannot open temporary file
	[説明]	中間ファイルをオープンできません。
	[対処方法]	HLNK_TMP 指定が正しいか確認してください。またはディスク容量に空きがない可能性があります。
F0563311	[メッセージ]	Cannot close temporary file
	[説明]	中間ファイルをクローズできません。ディスク容量に空きがない可能性があります。
F0563312	[メッセージ]	Cannot write temporary file
	[説明]	中間ファイルに書き込めません。ディスク容量に空きがない可能性があります。
F0563313	[メッセージ]	Cannot read temporary file
	[説明]	中間ファイルを読めません。ディスク容量に空きがない可能性があります。
F0563314	[メッセージ]	Cannot delete temporary file
	[説明]	中間ファイルを削除できません。ディスク容量に空きがない可能性があります。
F0563320	[メッセージ]	Memory overflow
	[説明]	最適化リンカが内部で使用するメモリが不足しています。
	[対処方法]	メモリを増やしてください。
F0563400	[メッセージ]	Cannot execute " *ロード・モジュール* "
	[説明]	" *ロード・モジュール* " を起動できません。
	[対処方法]	" *ロード・モジュール* " のパスが設定されているか確認してください。
F0563410	[メッセージ]	Interrupt by user
	[説明]	標準入力端末から「(Ctrl)+C」キーによる割り込みを検出しました。
F0563420	[メッセージ]	Error occurred in " *ロード・モジュール* "
	[説明]	" *ロード・モジュール* " 実行中にエラーが発生しました。
F0563430	[メッセージ]	The total section size exceeded the limit
	[説明]	無償評価版でリンク可能なサイズ制限を越えました。 【RH850】リンク・サイズを 256K バイト以内に制限しています。 【RX】リンク・サイズを 128K バイト以内に制限しています。 【RL78】リンク・サイズを 64K バイト以内に制限しています。
	[対処方法]	リンク・サイズを制限範囲内にしてください。または，製品版の購入をご検討ください。
F0578200	[メッセージ]	memory allocation fault
	[説明]	メモリが足りません。
F0578201	[メッセージ]	bad key *character* - use [dm(a|b)qr(a|b|u)txV]
	[説明]	*character* をキーとして指定することはできません。

F0578202	[メッセージ]	bad option *character* - use [cv]
	[説明]	*character* をオプションとして指定することはできません。
F0578203	[メッセージ]	bad option *string*
	[説明]	*string* をオプションとして指定することはできません。
F0578204	[メッセージ]	can not create file *file*
	[説明]	ファイル *file* を作成できません。
F0578205	[メッセージ]	file name *name*... is too long - limit is *number*
	[説明]	ファイル名 *name* の長さが限界を越えています。限界値は *number1* です。
F0578206	[メッセージ]	can not open file *file*
	[説明]	ファイル *file* をオープンできません。
F0578207	[メッセージ]	can not close file *file*
	[説明]	ファイル *file* をクローズできません。
F0578208	[メッセージ]	can not read file *file*
	[説明]	ファイル *file* からの読み込みができません。
F0578209	[メッセージ]	can not write file *file*
	[説明]	ファイル *file* への書き込みができません。
F0578210	[メッセージ]	can not seek file *file*
	[説明]	ファイル *file* をシークできません。
F0578212	[メッセージ]	can not nest command file *file*
	[説明]	コマンド・ファイル *file* がネストしています。ネストはできません。
F0578213	[メッセージ]	file is not library *file*
	[説明]	*file* はライブラリ・ファイルではありません。
F0578214	[メッセージ]	malformed library file *file*
	[説明]	ライブラリ・ファイル *file* の内容が破壊されているおそれがあります。
F0578215	[メッセージ]	can not find member *member*
	[説明]	ライブラリ・ファイル内にメンバ *member* が存在しません。
F0578216	[メッセージ]	symbol table limit error *file* (*number1*) - limit is *number2*
	[説明]	ライブラリ・ファイル *file* において，シンボルの個数 *number1* が限界を越えました。限界値は *number2* です。
F0578217	[メッセージ]	symbol table error *file*
	[説明]	ライブラリ・ファイル *file* において，ライブラリ・シンボル・テーブルの作成に失敗しました。
F0578218	[メッセージ]	string table error *file*
	[説明]	ライブラリ・ファイル *file* において，ライブラリ・ストリング・テーブルの内容が破壊されているおそれがあります。
F0578219	[メッセージ]	*file* has no member
	[説明]	ライブラリ・ファイル *file* 内にメンバが存在しません。

F0578220	［メッセージ］	version error *file*
	［説明］	指定されたファイル *file* の形式の版が本ライブラリアンの扱うことのできる版ではありません。
F0578221	［メッセージ］	can not read library header *file*
	［説明］	ライブラリ・ファイル *file* のヘッダの読み込みができません。
F0593113	［メッセージ］	Neither isa nor cpu is specified
	［説明］	-isa オプション，-cpu オプションのいずれの指定もありません。
F0593114	［メッセージ］	Both '-isa' option and '-cpu' option are specified
	［説明］	-isa オプションと -cpu オプションを同時に指定しています。
F0595001	［メッセージ］	ファイル " *ファイル名* " をオープンできません。
F0595002	［メッセージ］	ファイル " *ファイル名* " の読み込み中にエラーが生じました。
F0595003	［メッセージ］	ファイル " *ファイル名* " の書き込み中にエラーが生じました。
F0595004	［メッセージ］	ファイル " *ファイル名* " をクローズできません。

10.5.4 インフォメーション

表 10.5 インフォメーション

M0520009	[メッセージ]	Nested comment is not allowed.
	[説明]	コメントのネスティングは許されていません。
M0520018	[メッセージ]	Expected a ")".
	[説明]	")" がありません。
M0520111	[メッセージ]	Statement is unreachable.
	[説明]	文は実行されません。
M0520128	[メッセージ]	Loop is not reachable from preceding code.
	[説明]	ループはその前のコードから到達しません。
M0520174	[メッセージ]	Expression has no effect.
	[説明]	式は作用しません。
M0520193	[メッセージ]	Zero used for undefined preprocessing identifier *xxx*.
	[説明]	定義されていないプリプロセッサ識別子 *xxx* に対して 0 が使用されます。
M0520237	[メッセージ]	Selector expression is constant.
	[説明]	選択式が定数です。
M0520261	[メッセージ]	Access control not specified (" *名前* " by default).
	[説明]	基底クラスのアクセス制御指定がありません。アクセス制御指定 " *名前* " が仮定されます。
M0520324	[メッセージ]	Duplicate friend declaration.
	[説明]	フレンド宣言が重複して指定されています。
M0520381	[メッセージ]	Extra ";" ignored.
	[説明]	余分な ";" を無視しました。
M0520399	[メッセージ]	*名前* has an operator new *xxx*() but no default operator delete *xxx*().
	[説明]	" *名前* " が operator new を持ちますが，デフォルトの operator delete を持ちません。
M0520400	[メッセージ]	*名前* has a default operator delete *xxx*() but no operator new *xxx*().
	[説明]	" *名前* " がデフォルトの operator delete を持ちますが，operator new を持ちません。
M0520479	[メッセージ]	*名前* redeclared "inline" after being called.
	[説明]	関数が呼ばれたあとに inline" *名前* " を宣言しています。以降 inline 指定を有効にします。
M0520487	[メッセージ]	Inline *名前* cannot be explicitly instantiated.
	[説明]	インライン関数 " *名前* " を実体化することはできません。
M0520534	[メッセージ]	Use of a local type to specify an exception.
	[説明]	ローカルな型を使用した例外処理が指定されています。
M0520535	[メッセージ]	Redundant type in exception specification.
	[説明]	例外処理中に冗長な型の指定があります。

M0520549	[メッセージ]	*symbol* is used before its value is set.
	[説明]	*symbol* は値が設定される前に使用されました。
M0520618	[メッセージ]	Struct or union declares no named members.
	[説明]	構造体か共用体に名前のないメンバがあります。
M0520652	[メッセージ]	Calling convention is ignored for this type.
	[説明]	この型に対する呼び出し規約は無視されます。
M0520678	[メッセージ]	Call of "*symbol*" cannot be inlined.
	[説明]	*symbol* の呼び出しはインライン展開できません。
M0520679	[メッセージ]	*symbol* cannot be inlined.
	[説明]	*symbol* はインライン展開できません。
M0520815	[メッセージ]	Type qualifier on return type is meaningless.
	[説明]	返却型に対する型指定は意味がありません。
M0520831	[メッセージ]	Support for placement delete is disabled.
	[説明]	operator delete 関数の型が正しくありません。処理を継続します。
M0520863	[メッセージ]	Effect of this "#pragma pack" directive is local to *xxx*.
	[説明]	#pragma pack ディレクティブの影響はシンボル内にとどまります。
M0520866	[メッセージ]	Exception specification ignored.
	[説明]	例外指定は無視されます。
M0520949	[メッセージ]	Specifying a default argument on this declaration is nonstandard.
	[説明]	この宣言にデフォルト引数を指定するのは標準形式ではありません。
M0521348	[メッセージ]	Declaration hides "*symbol*".
	[説明]	宣言は "*symbol*" を隠します。
M0521353	[メッセージ]	*シンボル* has no corresponding member operator delete *xxx* (to be called if an exception is thrown during initialization of an allocated object).
	[説明]	" *シンボル* " は new オペレータの対となる delete オペレータを持ちません（取得したオブジェクトの初期化時に例外が発生した場合に呼ばれます）。
M0521380	[メッセージ]	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
	[説明]	仮想関数の定義がありません。また，無名空間のメンバであるため，それ以外の場所で定義することができません。
M0521381	[メッセージ]	Carriage return character in source line outside of comment or character/string literal.
	[説明]	改行文字がコメントまたは文字列リテラル以外のところにあります。
M0523009	[メッセージ]	This pragma has no effect.
	[説明]	この #pragma は無効です。
M0523028	[メッセージ]	Rule *ルール番号* : *内容*
	[説明]	MISRA-C:2004 の*ルール番号*と*内容*の該当箇所を検出しました。

M0523033	[メッセージ]	Precision lost
	[説明]	代入式において，右辺の式の値を左辺の型へ変換する時に，精度が失われる可能性があります。
M0560001	[メッセージ]	Section " *セクション* " created by optimization " *最適化* "
	[説明]	" *最適化* " の最適化によって，" セクション " を作成しました。
M0560002	[メッセージ]	Symbol " *シンボル* " created by optimization " *最適化* "
	[説明]	" *最適化* " の最適化によって，" シンボル " を作成しました。
M0560004	[メッセージ]	" *ファイル* "-" *シンボル* " deleted by optimization
	[説明]	symbol_delete の最適化によって，" *ファイル* " 内の " *シンボル* " を削除しました。
M0560005	[メッセージ]	The offset value from the symbol location has been changed by optimization : " *ファイル* "-" *セクション* "-" *シンボル* ± offset"
	[説明]	" *シンボル* ± offset" の範囲で最適化によるサイズ変更があったため offset 値を変更しました。問題ないか確認してください。offset 値の変更を抑止したい場合は，" *ファイル* " のアセンブル時に goptimize オプション指定を外してください。
M0560100	[メッセージ]	No inter-module optimization information in " *ファイル* "
	[説明]	" *ファイル* " 内にモジュール間最適化情報がありません。" *ファイル* " をモジュール間最適化の対象外にします。モジュール間最適化の対象にする場合は，コンパイル，アセンブル時に goptimize オプションを指定してください。
M0560101	[メッセージ]	No stack information in " *ファイル* "
	[説明]	" *ファイル* " 内にスタック情報がありません。" *ファイル* " はアセンブラ出力ファイルの可能性があります。最適化リンカが出力するスタック情報ファイルに当該ファイルの内容は含まれません。
M0560102	[メッセージ]	Stack size " *サイズ* " specified to the undefined symbol " *シンボル* " in " *ファイル* "
	[説明]	" *ファイル* " 内の未定義シンボル " *シンボル* " に，スタックサイズ " *サイズ* " が指定されています。
M0560103	[メッセージ]	Multiple stack sizes specified to the symbol " *シンボル* "
	[説明]	シンボル " *シンボル* " は，複数のスタックサイズが指定されています。
M0560300	[メッセージ]	Mode type " *モード種別 1* " in " *ファイル* " differ from " *モード種別 2* "
	[説明]	異なるモード種別のファイルを入力しました。
M0560400	[メッセージ]	Unused symbol " *ファイル* "-" *シンボル* "
	[説明]	" *ファイル* " 内の " *シンボル* " は使用されていません。
M0560500	[メッセージ]	Generated CRC code at " *アドレス* "
	[説明]	" *アドレス* " に CRC コードを出力しました。
M0560510	[メッセージ]	Section " *セクション* " was moved other area specified in option "cpu=< *メモリ属性* >"
	[説明]	セクションを分割せずに cpu=< *メモリ属性* > にしたがって " *セクション* " を配置しました。
M0560511	[メッセージ]	Sections " *セクション名* "," *分割後のセクション名* " are Non-contiguous
	[説明]	" *セクション名* " のセクションを分割し，" *分割後のセクション名* " のセクションを生成しました。

M0560512	［メッセージ］	Section " *セクション*" created by " *オプション*"
	［説明］	" *オプション*" によって，" *セクション*" を作成しました。
M0592150	［メッセージ］	*string1* を *string2* に変換しました。
M0592151	［メッセージ］	*string1* を削除しました。
M0592152	［メッセージ］	機能 を CX コンパイラ仕様に変換しました。
M0592153	［メッセージ］	I/O レジスタのビット指定アクセスをマクロ呼び出しに変換しました。
M0592154	［メッセージ］	I/O レジスタのビットアクセス用マクロを生成しました。
M0592155	［メッセージ］	#pragma interrupt に 'multi' オプションを追加しました。
M0592156	［メッセージ］	#pragma interrupt NO_VECT を生成しました。
M0592157	［メッセージ］	*string* を生成しました。
M0592250	［メッセージ］	*string1* を *string2* に変更しました。
M0592251	［メッセージ］	マルチステートメント ';' は改行へ変更しました。
M0592252	［メッセージ］	*string* を CX の形式に変更しました。
M0592253	［メッセージ］	*string1* を *string2* に変更しました。事前に .extern の指定も必要となるので確認してください。
	［説明］	.option data を $data，または .option sdata を $sdata へ変換しました。CX では，シンボルを事前に .extern で宣言する必要がありますので確認してください。
M0592270	［メッセージ］	*string* を削除しました。
M0592280	［メッセージ］	.previous は変更できません。適切なセクションを指定してください。
	［説明］	.previous は CX では未サポートです。
M0592281	［メッセージ］	*string* ' ビット幅 : 値 ' は変更できません。適切な値を *string* ' 値 ' の形式で指定してください。
	［説明］	string ' ビット幅 : 値 ' は，未サポートです。
M0592282	［メッセージ］	.option reg_mode を変更できません。適切なレジスタモードを指定してください。
	［説明］	.option reg_mode で指定した値が CX では未サポートです。
M0594201	［メッセージ］	ファイルを上書きしますか？
	［対処方法］	既存のファイルに対して上書き保存をします。 ［はい］で上書き保存します。 ［いいえ］で中止します。
M0594202	［メッセージ］	関数 (*関数名*) を初期値に戻しますか？
	［対処方法］	関数に対する設定を初期値に戻します。 ［はい］で初期値に戻します。 ［いいえ］で中止します。
M0594203	［メッセージ］	すべての関数を初期値に戻しますか？
	［対処方法］	すべての関数に対する設定を初期値に戻します。 ［はい］で初期値に戻します。 ［いいえ］で中止します。

10.5.5 ワーニング

表 10.6 ワーニング

W0511105	［メッセージ］	" *文字列* " オプションで指定された " *パス名* " はファイルです。フォルダを指定してください。
	［メッセージ］	【CC-RX】 "*path*" specified by the "*character string*" option is a file. Specify a folder.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された " *パス名* " はファイルです。フォルダを指定してください。
W0511106	［メッセージ］	" *文字列* " オプションで指定されたフォルダ " *フォルダ名* " が見つかりません。
	［メッセージ］	【CC-RX】 The folder "*folder*" specified by the "*character string*" option is not found.
	［説明］	【CC-RX】 " *文字列* " オプションで指定されたフォルダ " *フォルダ名* " が見つかりません。
W0511123	［メッセージ］	" *文字列 1"* オプションが指定されたので " *文字列 2*" オプションは無視しました。
	［メッセージ］	【CC-RX】 The "*character string2*" option is ignored when the "*character string1*" option is specified at the same time.
	［説明］	【CC-RX】 " *文字列 1*" オプションが指定されたので " *文字列 2*" オプションは無視しました。
W0511143	［メッセージ］	FPU を持っていないデバイスが指定されたので、"-Xfloat" オプションを無視しました。
	［メッセージ］	【CC-RX】 The "-Xfloat" option is ignored because specified device does not have FPU.
	［説明］	【CC-RX】 FPU を持っていないデバイスが指定されたので、"-Xfloat" オプションを無視しました。
W0511144	［メッセージ］	"-C" オプションと "-Xcommon" オプションが一致しません。" *文字列 1*" オプションの命令セットを無視します。" *文字列 2*" 命令セットのコモンオブジェクトを生成します。
	［メッセージ］	【CC-RX】 "-C" option and "-Xcommon" option is mismatch. Instruction set by "*character string1*" option is ignored. Create common object for "*character string2*" instruction set.
	［説明］	【CC-RX】 "-C" オプションと "-Xcommon" オプションが一致しません。" *文字列 1*" オプションの命令セットを無視します。" *文字列 2*" 命令セットのコモンオブジェクトを生成します。
W0511146	［メッセージ］	" *文字列* " オプションで指定された " *シンボル名* " はマクロでは使用できません。
	［メッセージ］	【CC-RX】 "*symbol name*" specified in the "*character string*" option is not allowed for a preprocessor macro.
	［説明］	【CC-RX】 " *文字列* " オプションで指定された " *シンボル名* " はマクロでは使用できません。
W0511147	［メッセージ］	" *文字列* " オプションが複数指定されています。後の指定が有効になります。
	［メッセージ］	【CC-RX】 The "*character string*" option is specified more than once. The latter is valid.
	［説明］	【CC-RX】 " *文字列* " オプションが複数指定されています。後の指定が有効になります。

W0511149	［メッセージ］	" *文字列 1*" オプションと " *文字列 2*" オプションが矛盾しています。" *文字列 2*" オプションを無視します。
	［メッセージ］	【CC-RX】 The "*character string2*" option is ignored when the "*character string1*" option and the "*character string2*" option are inconsistent.
	［説明］	【CC-RX】 " *文字列 1*" オプションと " *文字列 2*" オプションが矛盾しています。" *文字列 2*" オプションを無視します。
W0511151	［メッセージ］	" *文字列 1*" オプションが指定されていないので、" *文字列 2*" オプションを無視します。
	［メッセージ］	【CC-RX】 The "*character string2*" option is ignored when the "*character string1*" option is not specified.
	［説明］	【CC-RX】 " *文字列 1*" オプションが指定されていないので，" *文字列 2*" オプションを無視します。
W0511153	［メッセージ］	"-O *文字列*" が指定されたので最適化詳細オプションはクリアされました。最適化詳細オプションは "-O *文字列*" の後に指定してください。
	［メッセージ］	【CC-RX】 Optimization itemoptions were cleared when "-O*character string*" option is specified. Optimization itemoptions need to specify after "-O*character string*" option.
	［説明］	【CC-RX】 "-O *文字列*" が指定されたので最適化詳細オプションはクリアされました。最適化詳細オプションは "-O *文字列*" の後に指定してください。
W0511156	［メッセージ］	"-Xdev_path" オプションで指定されたフォルダ上に、デバイス・ファイルが見つかりません。
	［説明］	"-Xdev_path" オプションで指定されたフォルダ上に，デバイス・ファイルが見つかりません。標準のデバイス・ファイル・フォルダで探します。
	［メッセージ］	【CC-RX】 Device file is not found in the folder specified by the "-Xdev_path" option.
	［説明］	【CC-RX】 "-Xdev_path" オプションで指定されたフォルダ上に，デバイス・ファイルが見つかりません。
W0511164	［メッセージ］	【CC-RH】 同じファイル名 " *ファイル名* " が複数指定されています。
	［説明］	【CC-RH】 コマンドラインに同じファイル名が複数指定されています。 CC-RH は，同じファイル名を複数扱うことができません。 最後に指定されたもののみが有効になります。
	［メッセージ］	【CC-RX】 Dupulicate file name. "*file-name*".
	［説明］	【CC-RX】 同じファイル名 " *ファイル名* " が複数指定されています。
	［メッセージ］	【CC-RH】 同じファイル名 " *ファイル名* " が複数指定されています。
W0511166	［メッセージ］	"*macro name*" is not a valid predefined macro name.
	［説明］	マクロ名 " *マクロ名* " はプリディファインドマクロではありません。undefine オプションの指定を無効とします。
W0511168	［メッセージ］	"*option-name*" option has no effect in this version.
	［説明］	本バージョンでは無効なオプションです。
W0511169	［メッセージ］	" *内容* " is not valid in "*language specifications*"
	［説明］	表示された " *内容* " は " *言語仕様*（C，または C++）" では無効です。
W0511170	［メッセージ］	"*option-name*" option is ignored due to the specification of another option.
	［説明］	このオプションはほかのオプションの指定により無視されます。

W0511171	［メッセージ］	" *内容* " is ignored in "*language specifications*".
	［説明］	表示された " *内容* " は " *言語仕様*（C，または C++）" では無視されます。
W0511172	［メッセージ］	Nothing to compile, assemble or link.(input and output combination)
	［説明］	コンパイル，アセンブルまたはリンク処理のいずれも行う必要がありません。
	［対処方法］	入力ファイルの構成と output オプション指定の組み合わせを確認してください。
W0519999	［メッセージ］	The "*option-name*" option is not implemented.
	［説明］	" *オプション名* " オプションは実装されていません。
W0520001	［メッセージ］	ファイルの最終行が改行で終了していません。
	［対処方法］	改行を追加してください。
W0520009	［メッセージ］	コメントのネスティングは許されていません。
	［対処方法］	ネストしないようにしてください。
	［メッセージ］	【CC-RX】 Nested comment is not allowed.
	［説明］	【CC-RX】 コメントのネスティングは許されていません。
	［対処方法］	【CC-RX】 ネストしないようにしてください。
W0520011	［メッセージ］	不明な前処理指令があります。
	［メッセージ］	【CC-RX】 Unrecognized preprocessing directive.
	［説明］	【CC-RX】 不明な前処理指令があります。
W0520012	［メッセージ］	前に構文エラーがあるため、ここより文法の解析を再開します。
	［メッセージ］	【CC-RX】 Parsing restarts here after previous syntax error.
	［説明］	【CC-RX】 前に構文エラーがあるため，ここより文法の解析を再開します。
W0520014	［メッセージ］	前処理指令の後に不正な文字があります。
	［メッセージ］	【CC-RX】 Extra text after expected end of preprocessing directive.
	［説明］	【CC-RX】 前処理指令の後に不正な文字があります。
W0520019	［メッセージ］	Extra text after expected end of number.
	［説明］	数値の後に不正な文字があります。
W0520021	［メッセージ］	型修飾子はこの宣言では無効です。
	［説明］	型修飾子はこの宣言では無効です。無視しました。
	［メッセージ］	【CC-RX】 Type qualifiers are meaningless in this declaration.
	［説明］	【CC-RX】 型修飾子はこの宣言では無効です。無視しました。
W0520026	［メッセージ］	文字定数中の文字が多すぎます。
	［説明］	文字定数中の文字が多すぎます。文字定数は複数の文字を含むことはできません。
	［メッセージ］	【CC-RX】 Too many characters in character constant.
	［説明］	【CC-RX】 文字定数中の文字が多すぎます。文字定数は複数の文字を含むことはできません。
W0520027	［メッセージ］	char 型の値が範囲を超えています。
	［メッセージ］	【CC-RX】 char 型の値が範囲を越えています。
	［説明］	【CC-RX】 char 型の値が範囲を越えています。

W0520038	［メッセージ］	この前処理指令は許可されていません -- #else はすでにあります。
	［説明］	#else がすでにあるため，このディレクティブは不正です。
	［メッセージ］	【CC-RX】 Directive is not allowed -- an #else has already appeared.
	［説明］	【CC-RX】 この前処理指令は許可されていません -- #else はすでにあります。
W0520039	［メッセージ］	0 で除算を行いました。
	［メッセージ］	【CC-RX】 Division by zero.
	［説明］	【CC-RX】 0 で除算を行いました。
W0520042	［メッセージ］	オペランドの型が適合しません (" *型 1*" と " *型 2*")。
	［メッセージ］	【CC-RX】 Operand types are incompatible (" *型 1*" and " *型 2*").
	［説明］	【CC-RX】 オペランドの型が適合しません (" *型 1*" と " *型 2*")。
W0520045	［メッセージ］	#undef may not be used on this predefined name.
	［説明］	既定義名に対して #undef を使用できません。
W0520046	［メッセージ］	*symbol* is predefined; attempted redefinition ignored.
	［説明］	*symbol* を再定義することはできません。
W0520047	［メッセージ］	Incompatible redefinition of macro "*symbol*".
	［説明］	マクロ "*symbol*" の適合しない再定義があります。
W0520054	［メッセージ］	Too few arguments in macro invocation.
	［説明］	マクロに対する引数が足りません。
W0520055	［メッセージ］	Too many arguments in macro invocation.
	［説明］	マクロに対する引数が多すぎます。
W0520061	［メッセージ］	整数演算の結果が範囲を超えました。
	［メッセージ］	【CC-RX】 Integer operation result is out of range.
	［説明］	【CC-RX】 整数演算の結果が範囲を越えました。
W0520062	［メッセージ］	シフト数が負数です。
	［説明］	シフト数が負数です。ANSI-C では未定義の動作となります。
	［メッセージ］	【CC-RX】 Shift count is negative.
	［説明］	【CC-RX】 シフト数が負数です。ANSI-C では未定義の動作となります。
W0520063	［メッセージ］	シフト数が多すぎます。
	［メッセージ］	【CC-RX】 Shift count is too large.
	［説明］	【CC-RX】 シフト数が多すぎます。
W0520064	［メッセージ］	この宣言は何も宣言できません。
	［メッセージ］	【CC-RX】 Declaration does not declare anything.
	［説明］	【CC-RX】 この宣言は何も宣言できません。
W0520066	［メッセージ］	Enumeration value is out of "int" range.
	［説明］	enum の値が "int" の範囲を越えています。

W0520068	［メッセージ］	整数変換で結果の符号が反転しました。
	［メッセージ］	【CC-RX】 Integer conversion resulted in a change of sign.
	［説明］	【CC-RX】 整数変換で結果の符号が反転しました。
W0520069	［メッセージ］	整数変換で結果の値が丸められました。
	［メッセージ］	【CC-RX】 Integer conversion resulted in truncation.
	［説明］	【CC-RX】 整数変換で結果の値が丸められました。
W0520076	［メッセージ］	マクロに対する引数がありません。
	［メッセージ］	【CC-RX】 Argument to macro is empty.
	［説明］	【CC-RX】 マクロに対する引数がありません。
W0520077	［メッセージ］	宣言に記憶域クラスまたは型指定子がありません。
	［メッセージ］	【CC-RX】 This declaration has no storage class or type specifier.
	［説明］	【CC-RX】 宣言に記憶域クラスまたは型指定子がありません。
W0520082	［メッセージ］	記憶域クラスが最初にありません。
	［説明］	記憶域クラスが最初にありません。記憶域クラスは宣言の最初に指定してください。
	［メッセージ］	【CC-RX】 Storage class is not first.
	［説明］	【CC-RX】 記憶域クラスが最初にありません。記憶域クラスは宣言の最初に指定してください。
W0520083	［メッセージ］	Type qualifier specified more than once.
	［説明］	型修飾子が複数回指定されました。
W0520085	［メッセージ］	Invalid storage class for a parameter.
	［説明］	引数に対する記憶域クラスが不正です。
W0520086	［メッセージ］	Invalid storage class for a function.
	［説明］	関数に対する記憶域クラスが不正です。
W0520099	［メッセージ］	ここでの宣言は引数宣言でなければなりません。
	［メッセージ］	【CC-RX】 A declaration here must declare a parameter.
	［説明］	【CC-RX】 ここでの宣言は引数宣言でなければなりません。
W0520101	［メッセージ］	*xxx* has already been declared in the current scope.
	［説明］	*xxx* はすでにこのスコープで宣言されています。
W0520108	［メッセージ］	1 ビットの符号付きビット・フィールドです。
	［メッセージ］	【CC-RX】 Signed bit field of length 1.
	［説明］	【CC-RX】 1 ビットの符号付きビット・フィールドです。
W0520111	［メッセージ］	文は実行されません。
	［メッセージ］	【CC-RX】 Statement is unreachable.
	［説明］	【CC-RX】 文は実行されません。

W0520117	[メッセージ]	void でない関数 " *関数名* " は値を返す必要があります。
	[メッセージ]	【CC-RX】 Non-void "*symbol*" should return a value.
	[説明]	【CC-RX】 void でない関数 " *関数名* " は値を返す必要があります。
W0520118	[メッセージ]	A void function may not return a value.
	[説明]	void 関数は値を返せません。
W0520127	[メッセージ]	文がありません。
	[メッセージ]	【CC-RX】 Expected a statement.
	[説明]	【CC-RX】 文がありません。
W0520128	[メッセージ]	ループはその前のコードから到達しません。
	[メッセージ]	【CC-RX】 Loop is not reachable from preceding code.
	[説明]	【CC-RX】 ループはその前のコードから到達しません。
W0520138	[メッセージ]	レジスタ変数に対するアドレス演算子は許されていません。
	[メッセージ]	【CC-RX】 Taking the address of a register variable is not allowed.
	[説明]	【CC-RX】 レジスタ変数に対するアドレス演算子は許されていません。
W0520139	[メッセージ]	Taking the address of a bit field is not allowed.
	[説明]	ビット・フィールドに対するアドレス演算子は許されていません。
W0520140	[メッセージ]	Too many arguments in function call.
	[説明]	関数呼び出しに対する引数が多すぎます。
W0520144	[メッセージ]	型 " *型名 1*" の値は型 " *型名 2*" の実体の初期化には使用できません。
	[メッセージ]	【CC-RX】 A value of type " *型名 1*"" cannot be used to initialize an entity of type " *型名 2*".
	[説明]	【CC-RX】 型 " *型名 1*" の値は型 " *型名 2*" の実体の初期化には使用できません。
W0520147	[メッセージ]	宣言は " *宣言* "(宣言位置 行 *行番号*) と整合しません。
	[メッセージ]	【CC-RX】 Declaration is incompatible with " *宣言* "(declared at line *行番号*).
	[説明]	【CC-RX】 宣言は " *宣言* " と整合しません。
W0520152	[メッセージ]	0 でない値がポインタに変換されました。
	[メッセージ]	【CC-RX】 Conversion of nonzero integer to pointer.
	[説明]	【CC-RX】 0 でない値がポインタに変換されました。
W0520157	[メッセージ]	Expression must be an integral constant expression.
	[説明]	式は整数定数式である必要があります。
W0520161	[メッセージ]	認識されない #pragma です。
	[メッセージ]	【CC-RX】 Unrecognized #pragma.
	[説明]	【CC-RX】 認識されない #pragma です。
W0520165	[メッセージ]	関数呼び出しに引数が足りません。
	[メッセージ]	【CC-RX】 Too few arguments in function call.
	[説明]	【CC-RX】 関数呼び出しに引数が足りません。

W0520167	［メッセージ］	" *型名 1*" 型の引数は型 " *型名 2*" の引数と整合しません。
	［メッセージ］	【CC-RX】 Argument of type " *型名 1*" is incompatible with parameter of type " *型名 2*".
	［説明］	【CC-RX】 " *型名 1*" 型の引数は型 " *型名 2*" の引数と整合しません。
W0520170	［メッセージ］	ポインタがオブジェクトから外れた位置を指しました。
	［メッセージ］	【CC-RX】 Pointer points outside of underlying object.
	［説明］	【CC-RX】 ポインタがオブジェクトから外れた位置を指しました。
W0520172	［メッセージ］	外部または内部リンケージが以前の宣言と整合しません。
	［メッセージ］	【CC-RX】 External/internal linkage conflict with previous declaration.
	［説明］	【CC-RX】 外部または内部リンケージが以前の宣言と整合しません。
W0520173	［メッセージ］	浮動小数点数は要求された整数型に入りません。
	［メッセージ］	【CC-RX】 Floating-point value does not fit in required integral type.
	［説明］	【CC-RX】 浮動小数点数は要求された整数型に入りません。
W0520174	［メッセージ］	式は作用しません。
	［説明］	式は作用しません。無効です。
	［メッセージ］	【CC-RX】 Expression has no effect.
	［説明］	【CC-RX】 式は作用しません。無効です。
W0520175	［メッセージ］	添字が範囲を越えました。
	［メッセージ］	【CC-RX】 Subscript out of range.
	［説明］	【CC-RX】 添字が範囲を越えました。
W0520177	［メッセージ］	*種別*" *シンボル名*" は宣言されましたが参照されていません。
	［メッセージ］	【CC-RX】 *種別*" *シンボル名*" was declared but never referenced
	［説明］	【CC-RX】 *種別*" *シンボル名*" は宣言されましたが参照されていません。
W0520179	［メッセージ］	"%" の右オペランドが 0 です。
	［メッセージ］	【CC-RX】 Right operand of "%" is zero.
	［説明］	【CC-RX】 "%" の右オペランドが 0 です。
W0520180	［メッセージ］	実引数が仮引数と整合しません。
	［メッセージ］	【CC-RX】 Argument is incompatible with formal parameter.
	［説明］	【CC-RX】 実引数が仮引数と整合しません。
W0520181	［メッセージ］	引数は対応するフォーマット文字列に整合しません。
W0520185	［メッセージ］	実行されないコードの中に動的初期化があります。
	［メッセージ］	【CC-RX】 Dynamic initialization in unreachable code.
	［説明］	【CC-RX】 実行されないコードの中に動的初期化があります。
W0520186	［メッセージ］	符号なし整数と 0 の比較は無意味です。
	［メッセージ］	【CC-RX】 Pointless comparison of unsigned integer with zero.
	［説明］	【CC-RX】 符号なし整数と 0 の比較は無意味です。

W0520187	［メッセージ］	"==" と思われる "=" の使用があります。
	［メッセージ］	【CC-RX】 Use of "=" where "==" may have been intended.
	［説明］	【CC-RX】 "==" と思われる "=" の使用があります。
W0520188	［メッセージ］	列挙型に別の型が混在しています。
	［メッセージ］	【CC-RX】 Enumerated type mixed with another type.
	［説明］	【CC-RX】 列挙型に別の型が混在しています。
W0520191	［メッセージ］	型修飾子はキャスト型に意味を持ちません。
	［メッセージ］	【CC-RX】 Type qualifier is meaningless on cast type.
	［説明］	【CC-RX】 型修飾子はキャスト型に意味を持ちません。
W0520192	［メッセージ］	認識されないエスケープ・シーケンスがあります。
	［メッセージ］	【CC-RX】 Unrecognized character escape sequence.
	［説明］	【CC-RX】 認識されないエスケープ・シーケンスがあります。
W0520223	［メッセージ］	Function *xxx* declared implicitly.
	［説明］	関数 *xxx* は暗黙に宣言されました。
W0520224	［メッセージ］	フォーマット文字列にはさらなる引数が必要です。
	［メッセージ］	【CC-RX】 The format string requires additional arguments.
	［説明］	【CC-RX】 フォーマット文字列にはさらなる引数が必要です。
W0520225	［メッセージ］	フォーマット文字列に対して引数が多すぎます。
	［メッセージ］	【CC-RX】 The format string ends before this argument.
	［説明］	【CC-RX】 フォーマット文字列に対して引数が多すぎます。
W0520226	［メッセージ］	不正なフォーマット文字列です。
	［メッセージ］	【CC-RX】 Invalid format string conversion.
	［説明］	【CC-RX】 不正なフォーマット文字列です。
W0520229	［メッセージ］	ビット・フィールドは列挙型のすべての値を保持できません。
	［メッセージ］	【CC-RX】 Bt field cannot contain all values of the enumerated type.
	［説明］	【CC-RX】 ビット・フィールドは列挙型のすべての値を保持できません。
W0520231	［メッセージ］	宣言は関数の外で見えません。
	［メッセージ］	【CC-RX】 Declaration is not visible outside of function.
	［説明］	【CC-RX】 宣言は関数の外で見えません。
W0520232	［メッセージ］	Old-fashioned typedef of "void" ignored.
	［説明］	古い仕様の "void" typedef です。無視されました。
W0520236	［メッセージ］	制御式が定数です。
	［メッセージ］	【CC-RX】 Controlling expression is constant.
	［説明］	【CC-RX】 制御式が定数です。
W0520257	［メッセージ］	const "*symbol*" requires an initializer.
	［説明］	const 変数 "*symbol*" は初期化子が必要です。

W0520260	[メッセージ]	明示的な型がありません。"int" として扱います。
	[メッセージ]	【CC-RX】 Explicit type is missing ("int" assumed).
	[説明]	【CC-RX】 明示的な型がありません。"int" として扱います。
W0520262	[メッセージ]	Not a class or struct name.
	[説明]	基底クラスで指定されたクラスまたは構造体がありません。
W0520280	[メッセージ]	Declaration of a member with the same name as its class.
	[説明]	クラス名と同じ名前のメンバ名を宣言しています。
W0520284	[メッセージ]	NULL reference is not allowed.
	[説明]	NULL へのリファレンスは許されません。指定された通りに式を評価します。
W0520296	[メッセージ]	Invalid use of non-lvalue array.
	[説明]	左辺値でない配列の不正な利用です。
W0520300	[メッセージ]	A pointer to a bound function may only be used to call the function.
	[説明]	メンバ関数へのポインタを関数呼び出し以外に使用しています。
W0520301	[メッセージ]	typedef 名はすでに同じ型で宣言されています。
	[メッセージ]	【CC-RX】 typedef name has already been declared (with same type).
	[説明]	【CC-RX】 typedef 名はすでに同じ型で宣言されています。
W0520326	[メッセージ]	Inline is not allowed.
	[説明]	inline は許されていません。
W0520335	[メッセージ]	Linkage specification is not allowed.
	[説明]	リンケージの指定は許されていません。
W0520368	[メッセージ]	*xxx* defines no constructor to initialize the following:
	[説明]	名前は初期化のためのコンストラクタを定義していません。
W0520370	[メッセージ]	*symbol* has an uninitialized const field.
	[説明]	*symbol* は const のフィールドが初期化されていません。
W0520375	[メッセージ]	宣言は typedef 名を必要とします。
	[メッセージ]	【CC-RX】 Declaration requires a typedef name.
	[説明]	【CC-RX】 宣言は typedef 名を必要とします。
W0520377	[メッセージ]	"virtual" is not allowed.
	[説明]	virtual を指定することはできません。
W0520381	[メッセージ]	Extra ";" ignored.
	[説明]	余分な ";" を無視しました。
W0520382	[メッセージ]	In-class initializer for nonstatic member is nonstandard.
	[説明]	非スタティックなメンバを初期化するのは標準形式ではありません。
W0520414	[メッセージ]	Delete of pointer to incomplete class.
	[説明]	不完全型クラスへのポインタは削除されました。

W0520430	[メッセージ]	Returning reference to local temporary.
	[説明]	関数内にローカルな領域のリファレンスをリターン値にしています。
W0520494	[メッセージ]	typedef を伴う void の引数リストの宣言は標準でありません。
	[メッセージ]	【CC-RX】 Declaring a void parameter list with a typedef is nonstandard.
	[説明]	【CC-RX】 typedef を伴う void の引数リストの宣言は標準でありません。
W0520497	[メッセージ]	Declaration of %sq hides template parameter.
	[説明]	名前の宣言はテンプレート引数を隠蔽します。
W0520512	[メッセージ]	Type qualifier on a reference type is not allowed.
	[説明]	リファレンス型に const/volatile 修飾を指定することはできません。
W0520513	[メッセージ]	A value of type "*type1*" cannot be assigned to an entity of type "*type2*".
	[説明]	型 "*type1*" の値は型 "*type2*" の実体として代入できません。
W0520514	[メッセージ]	負の定数と unsigned 型の比較は意味がありません。
	[メッセージ]	【CC-RX】 Pointless comparison of unsigned integer with a negative constant.
	[説明]	【CC-RX】 負の定数と unsigned 型の比較は意味がありません。
W0520520	[メッセージ]	集成体は "{...}" により初期化してください。
	[メッセージ]	【CC-RX】 Initialization with "{...}" expected for aggregate object.
	[説明]	【CC-RX】 集成体は "{...}" により初期化してください。
W0520522	[メッセージ]	Pointless friend declaration.
	[説明]	自分自身へのフレンド宣言をしています。
W0520523	[メッセージ]	"." used in place of "::" to form a qualified name.
	[説明]	“.” がスコープ解決子 “::” の代わりに使用されています。
W0520533	[メッセージ]	Handler is potentially masked by previous handler for type "type".
	[説明]	ハンドラは型 "type" の以前のハンドラに潜在的にマスクされます。
W0520541	[メッセージ]	Omission of exception specification is incompatible with previous *名前*.
	[説明]	例外処理の省略形が前の " *名前* " と合致しません。
W0520546	[メッセージ]	初期化されないパスがあります。 : *種別* " *シンボル名* "(宣言位置 行 *行番号*)
W0520549	[メッセージ]	*種別* " *シンボル名* " は値が設定される前に使用されました。
	[メッセージ]	【CC-RX】 "*symbol*" is used before its value is set.
	[説明]	【CC-RX】 " *シンボル名* " は値が設定される前に使用されました。
W0520550	[メッセージ]	*種別* " *シンボル名* " は設定されていますが利用されていません。
	[メッセージ]	【CC-RX】 "*symbol*" was set but never used.
	[説明]	【CC-RX】 " *シンボル名* " は設定されていますが利用されていません。
W0520552	[メッセージ]	Exception specification is not allowed.
	[説明]	例外処理指定は許されません。例外処理を無効にします。
W0520553	[メッセージ]	external/internal linkage conflict for "*symbol*".
	[説明]	external/internal のリンケージが不適合を起こしています "*symbol*"。

W0520554	[メッセージ]	*名前* will not be called for implicit or explicit conversions.
	[説明]	変換関数 " *名前* " は暗黙的にも明示的にも呼ばれることはありません。
W0520611	[メッセージ]	Overloaded virtual function *名前 1* is only partially overridden in *名前 2*.
	[説明]	*名前 1* のオーバーロード仮想関数は " *名前 2*" の中で一部の仮想関数だけが置き換えの対象になります。指定された通りに処理を継続します。
W0520614	[メッセージ]	診断制御オプションに無効なエラー番号があります : 。
W0520617	[メッセージ]	Pointer-to-member-function cast to pointer to function.
	[説明]	メンバ関数ポインタを関数ポインタにキャストしています。
W0520618	[メッセージ]	構造体か共用体に名前のないメンバがあります。
	[メッセージ]	【CC-RX】 Struct or union declares no named members.
	[説明]	【CC-RX】 構造体か共用体に名前のないメンバがあります。
W0520650	[メッセージ]	Calling convention specified here is ignored.
	[説明]	ここでの呼び出し規則指定は無視されました。
W0520657	[メッセージ]	Inline specification is incompatible with previous "*symbol*".
	[説明]	inline 指定子は前の "*symbol*" と整合しません。
W0520662	[メッセージ]	Call of pure virtual function.
	[説明]	純粋仮想関数が関数を呼び出しています。
W0520676	[メッセージ]	*種別* " *シンボル名* "(宣言位置 行 *行番号*) の宣言のスコープ外で使用されました。
	[メッセージ]	【CC-RX】 Using out-of-scope declaration of *種別* " *シンボル名* "(declared at line *行番号*).
	[説明]	【CC-RX】 " *シンボル名* " の宣言のスコープ外で使用されました。
W0520691	[メッセージ]	*xxx*, required for copy that was eliminated, is inaccessible.
	[説明]	コピーコンストラクタにアクセスできません。
W0520692	[メッセージ]	*xxx*, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
	[説明]	コピーコンストラクタを呼び出すことができません。
W0520708	[メッセージ]	Incrementing a bool value is deprecated.
	[説明]	bool 型の値をインクリメントしています。値をインクリメントして処理を継続します。
W0520720	[メッセージ]	Redeclaration of *xxx* is not allowed to alter its access.
	[説明]	名前の再宣言でアクセス指定を変更することはできません。前の宣言のアクセス指定を有効にします。
W0520722	[メッセージ]	Use of alternative token "<:" appears to be unintended.
	[説明]	2 文字表記 "<:" が使用されました。"[" と解釈します。
W0520723	[メッセージ]	Use of alternative token "%%:" appears to be unintended.
	[説明]	2 文字表記 "%:" が使用されました。"#" と解釈します。
W0520737	[メッセージ]	Using-declaration ignored -- it refers to the current namespace.
	[説明]	現在の namespace スコープの名前を using 宣言しています。using 宣言を無視します。

W0520748	[メッセージ]	Calling convention specified more than once.
	[説明]	呼び出し規則指定が複数回指定されています。
W0520760	[メッセージ]	*シンボル* explicitly instantiated more than once.
	[説明]	" *シンボル* " を具現化できませんでした。
W0520767	[メッセージ]	Conversion from pointer to smaller integer.
	[説明]	ポインタが幅の小さな整数に変換されました。
W0520780	[メッセージ]	Reference is to *シンボル 1* -- under old for-init scoping rules it would have been *シンボル 2*.
	[説明]	" *シンボル 1*" を参照しています。
W0520783	[メッセージ]	Empty comment interpreted as token-pasting operator "##".
	[説明]	空のコメントは字句連結オペレータ "##" と仮定します。
W0520794	[メッセージ]	Template parameter %sq may not be used in an elaborated type specifier.
	[説明]	class 指定にテンプレート引数を使用することはできません。class 指定を無効にしてテンプレートを有効にします。
W0520802	[メッセージ]	Specifying a default argument when redeclaring an unreferenced function template is nonstandard.
	[説明]	未参照の関数テンプレートを再宣言するときにデフォルト引数を指定しています。デフォルト引数を無視します。
W0520806	[メッセージ]	Omission of exception specification is incompatible with *名前*.
	[説明]	throw 例外指定の省略は " *名前* " の例外指定と合致しません。" *名前* " を有効にします。
W0520812	[メッセージ]	const object requires an initializer -- class *型* has no explicitly declared default constructor.
	[説明]	const 型オブジェクトには初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。
W0520815	[メッセージ]	返却型に対する型修飾子は意味がありません。
	[メッセージ]	【CC-RX】 Type qualifier on return type is meaningless.
	[説明]	【CC-RX】 返却型に対する型修飾子は意味がありません。
W0520825	[メッセージ]	Virtual inline *名前* was never defined.
	[説明]	仮想インラインメンバ関数 " *名前* " の定義がありません。
W0520826	[メッセージ]	*名前* was never referenced.
	[説明]	関数の引数 " *名前* " は参照されません。
W0520829	[メッセージ]	Double used for "long double" in generated C code.
	[説明]	double が "long double" として生成された C コードで使用されています。
W0520830	[メッセージ]	*xxx* has no corresponding operator delete*yyy* (to be called if an exception is thrown during initialization of an allocated object).
	[説明]	対応する operator delete がありません。
W0520831	[メッセージ]	Support for placement delete is disabled
	[説明]	operator delete 関数の型が正しくありません。

W0520836	[メッセージ]	Returning reference to local variable.
	[説明]	局所変数のリファレンスをリターン値に指定しています。指定された処理を継続します。
W0520837	[メッセージ]	Omission of explicit type is nonstandard ("int" assumed).
	[説明]	型指定がありません。int 型を仮定します。
W0520867	[メッセージ]	Declaration of "size_t" does not match the expected type "*type*".
	[説明]	size_t の宣言は期待された型 "*type*" と一致しません。
W0520870	[メッセージ]	不正な多バイト文字列です。
	[メッセージ]	【CC-RX】 Invalid multibyte character sequence.
	[説明]	【CC-RX】 不正な多バイト文字列です。
W0520902	[メッセージ]	型修飾子を無視しました。
	[メッセージ]	【CC-RX】 Type qualifier ignored.
	[説明]	【CC-RX】 型修飾子を無視しました。
W0520912	[メッセージ]	Ambiguous class member reference -- *シンボル 1* used in preference to *シンボル 2*.
	[説明]	あいまいなクラスメンバの参照です。*シンボル 1* を *シンボル 2* に優先して参照します。
W0520925	[メッセージ]	Type qualifiers on function types are ignored.
	[説明]	関数型の型修飾子は無視されました。
W0520936	[メッセージ]	Redeclaration of *名前* alters its access.
	[説明]	" *名前* " の再宣言でアクセス指定を変更しています。再定義されたアクセス指定を有効にします。
W0520940	[メッセージ]	Missing return statement at end of non-void "*symbol*"..
	[説明]	void でない "*symbol*" に return 文がありません。
W0520941	[メッセージ]	Duplicate using-declaration of *名前* ignored.
	[説明]	using 宣言 " *名前* " を重複指定しています。重複した using 宣言を無効にします。
W0520942	[メッセージ]	enum bit-fields are always unsigned, but enum *xxx* includes negative enumerator.
	[説明]	列挙型のビットフィールドは常に unsigned です。しかし enum "type" は負の列挙子を持ちます。
W0520948	[メッセージ]	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
	[説明]	非標準形式のローカルクラスのフレンド宣言です。クラスの定義内に前方宣言がありません。
W0520951	[メッセージ]	Return type of function "main" must be "int".
	[説明]	main 関数の返却型は "int" である必要があります。
W0520959	[メッセージ]	Declared size for bit field is larger than the size of the bit field type; truncated to any-string bits.
	[説明]	宣言されたビット・フィールドの幅はビット・フィールドの型より大きいです。any-string ビットに縮小されます。
W0520961	[メッセージ]	Use of a type with no linkage to declare a variable with linkage.
	[説明]	リンケージを持たない型がリンケージを持つ変数宣言に使用されました。

W0520962	[メッセージ]	Use of a type with no linkage to declare a function.
	[説明]	リンケージを持たない型が関数に使用されました。
W0520970	[メッセージ]	The qualifier on this friend declaration is ignored.
	[説明]	このフレンド宣言への修飾子は無視されます。
W0520973	[メッセージ]	Inline used as a function qualifier is ignored.
	[説明]	inline 関数修飾子は無視されました。
W0520984	[メッセージ]	operator new and operator delete cannot be given internal linkage.
	[説明]	operator new/operator delete が static で定義されています。
W0520991	[メッセージ]	Extra braces are nonstandard.
	[説明]	余分な括弧は標準ではありません。
W0520993	[メッセージ]	" *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
	[メッセージ]	【CC-RX】 Subtraction of pointer types " *型名 1*" and " *型名 2*" is nonstandard.
	[説明]	【CC-RX】 " *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
W0520997	[メッセージ]	*関数名 2* is hidden by *関数名 1* -- virtual function override intended?
	[説明]	" *関数名 1*" が " *関数名 2*" を隠しています。仮想関数をオーバーライドしようとしていないか確認してください。
W0521000	[メッセージ]	記憶域クラスはここでは指定できません。
	[メッセージ]	【CC-RX】 A storage class may not be specified here.
	[説明]	【CC-RX】 記憶域クラスはここでは指定できません。
W0521028	[メッセージ]	Invalid redeclaration of nested class.
	[説明]	クラス内でクラスを二重定義しています。
W0521030	[メッセージ]	A variable with static storage duration cannot be defined within an inline function.
	[説明]	静的変数は inline 関数で定義できません。
W0521046	[メッセージ]	Floating-point value cannot be represented exactly.
	[説明]	浮動小数点の値が正しく記述されていません。
W0521050	[メッセージ]	Imaginary *= imaginary sets the left-hand operand to zero.
	[説明]	虚数 *= 虚数は左オペランドを 0 にします。
W0521051	[メッセージ]	Standard requires that "*symbol*" be given a type by a subsequent declaration ("int" assumed).
	[説明]	*symbol* の型は直後の宣言により与えられなければなりません ("int" に仮定されます)。
W0521053	[メッセージ]	整数がそれより幅の小さなポインタに変換されました。
	[メッセージ]	【CC-RX】 Conversion from integer to smaller pointer.
	[説明]	【CC-RX】 整数がそれより幅の小さなポインタに変換されました。
W0521055	[メッセージ]	Types cannot be declared in anonymous unions.
	[説明]	型を無名共用体内で宣言することはできません。

W0521056	[メッセージ]	ローカル変数へのポインタが返却されました。
	[メッセージ]	【CC-RX】 Returning pointer to local variable.
	[説明]	【CC-RX】 ローカル変数へのポインタが返却されました。
W0521057	[メッセージ]	Returning pointer to local temporary.
	[説明]	ローカルな領域へのポインタを返しています。
W0521072	[メッセージ]	A declaration cannot have a label.
	[説明]	宣言はラベルを持てません。
W0521105	[メッセージ]	#warning 指令 : *文字列*。
	[説明]	ソース・ファイル中に #warning 指令がありました。
	[メッセージ]	【CC-RX】 #warning directive: *文字列* .
	[説明]	【CC-RX】 #warning 指令 : *文字列*。
W0521145	[メッセージ]	*型 1* would have been promoted to " *型 2*" when passed through the ellipsis parameter; use the latter type instead.
	[説明]	*型 1* は*型 2* へと拡張されます。*型 2* を使用します。
W0521163	[メッセージ]	va_start should only appear in a function with an ellipsis parameter.
	[説明]	va_start が使用されるのは省略記号を引数とする関数のみです。
W0521192	[メッセージ]	Null (zero) character in input line ignored.
	[説明]	null 文字 (zero) の入力は無視されました。
W0521193	[メッセージ]	Null (zero) character in string or character constant.
	[説明]	null 文字 (zero) が文字列か文字定数にありました。
W0521194	[メッセージ]	Null (zero) character in header name.
	[説明]	null 文字 (zero) がヘッダ・ファイル名にあります。
W0521197	[メッセージ]	The prototype declaration of %nfd is ignored after this unprototyped redeclaration.
	[説明]	関数原型を無視します。
W0521211	[メッセージ]	Nonstandard cast to array type ignored.
	[説明]	標準でない配列型へのキャストは無視されました。
W0521213	[メッセージ]	field uses tail padding of a base class.
	[説明]	フィールドは基底クラスの終端パディングを使用しています。
W0521218	[メッセージ]	Base class *xxx* uses tail padding of base class *yyy*.
	[説明]	基底クラス 1 は基底クラス 2 の終端パディングを使用しています。
W0521222	[メッセージ]	Invalid error number.
	[説明]	不正なエラー番号です。
W0521223	[メッセージ]	Invalid error tag.
	[説明]	不正なエラータグです。
W0521224	[メッセージ]	Expected an error number or error tag.
	[説明]	エラー番号かエラータグがありません。

W0521235	[メッセージ]	Nonstandard conversion between pointer to function and pointer to data.
	[説明]	非標準形式の変換がポインタ関数と不完全なオブジェクト間で行われました。
W0521273	[メッセージ]	Alignment-of operator applied to incomplete type.
	[説明]	オペレータの整列条件が不完全な型に対して適用されました。
W0521285	[メッセージ]	Nonstandard qualified name in namespace member declaration.
	[説明]	非標準形式の修飾子名が名前空間のメンバの宣言に使用されています。
W0521290	[メッセージ]	Non-POD class type passed through ellipsis.
	[説明]	非 POD クラス型が省略記号に渡されています。
W0521294	[メッセージ]	Integer operand may cause fixed-point overflow.
	[説明]	整数オペランドは固定小数点でオーバフローを起こす原因となります。
W0521296	[メッセージ]	Fixed-point value cannot be represented exactly.
	[説明]	固定小数点値が正確に表現できません。
W0521297	[メッセージ]	Constant is too large for long long; given unsigned long long type (nonstandard).
	[説明]	定数が long long 型には大きすぎます。unsigned long long 型にしてください（標準ではありません）。
W0521301	[メッセージ]	*xxx* declares a non-template function -- add <> to refer to a template instance.
	[説明]	非テンプレート関数を宣言しています。
W0521302	[メッセージ]	Operation may cause fixed-point overflow.
	[説明]	処理は固定小数点のオーバフローを起こす原因となります。
W0521307	[メッセージ]	Class member typedef may not be redeclared.
	[説明]	クラスメンバの typedef を再宣言してはいけません。
W0521308	[メッセージ]	Taking the address of a temporary.
	[説明]	テンポラリのアドレスが取得されました。
W0521310	[メッセージ]	Fixed-point value implicitly converted to floating-point type.
	[説明]	固定小数点値が暗黙に浮動小数点型に変換されました。
W0521316	[メッセージ]	Value cannot be converted to fixed-point value exactly.
	[説明]	値は厳密に固定小数点値に変換できません。
W0521319	[メッセージ]	Fixed-point operation result is out of range.
	[説明]	固定小数点の処理が範囲を越えました。
W0521342	[メッセージ]	const_cast to enum type is nonstandard.
	[説明]	const_cast で列挙型をキャストするのは標準形式ではありません。
W0521346	[メッセージ]	Function returns incomplete class type %t.
	[説明]	関数が不正なクラス型を返しています。
W0521361	[メッセージ]	Negation of an unsigned fixed-point value.
	[説明]	符号なし固定小数点値を反転しました。

W0521373	[メッセージ]	Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
	[説明]	64 ビット整数型がより小さい整数型へと暗黙的に変換されています。移植性の問題になる可能性があります。
W0521374	[メッセージ]	Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
	[説明]	64 ビット整数型がより小さい整数型へと明示的に変換されています。移植性の問題になる可能性があります。
W0521375	[メッセージ]	Conversion from pointer to same-sized integral type (potential portability problem).
	[説明]	ポインタが同じサイズの整数型に変換されました（潜在的な移植性の問題があります）。
W0521386	[メッセージ]	Storage specifier ignored.
	[説明]	記憶クラス指定子を無視します。
W0521396	[メッセージ]	連結行のバック・スラッシュと改行の間に空白類がありました。無視します。
	[メッセージ]	【CC-RX】 White space between backslash and newline in line splice ignored.
	[説明]	【CC-RX】 連結行のバックスラッシュと改行の間に空白類がありました。無視します。
W0521400	[メッセージ]	positional format specifier cannot be zero.
	[説明]	positional format 指定子は 0 を指定できません。
W0521420	[メッセージ]	Some enumerator values cannot be represented by the integral type underlying the enum type.
	[説明]	いくつかの列挙子はその列挙型の潜在的な整数型で表現できません。
W0521422	[メッセージ]	多バイト文字リテラルです。潜在的な移植性の問題があります。
	[メッセージ]	【CC-RX】 Multicharacter character literal (potential portability problem).
	[説明]	【CC-RX】 多バイト文字リテラルです。潜在的な移植性の問題があります。
W0521427	[メッセージ]	offsetof applied to non-POD types is nonstandard.
	[説明]	offsetof を non-POD 型に指定するのは標準ではありません。
W0521433	[メッセージ]	No prior push_macro for *xxx*.
	[説明]	*xxx* の push_macro は優先されません。
W0521443	[メッセージ]	__real/__imag applied to real value.
	[説明]	__real/__imag が実数に指定されました。
W0521444	[メッセージ]	*symbol* was declared "deprecated (*xxx*)".
	[説明]	*symbol* は "deprecated (*xxx*)" に宣言されました。
W0521546	[メッセージ]	Argument must be a constant null pointer value.
	[説明]	引数は null ポインタ値でなければなりません。
W0521547	[メッセージ]	Insufficient number of arguments for sentinel value.
	[説明]	センチネル値への引数番号は不適当です。
W0521548	[メッセージ]	Sentinel argument must correspond to an ellipsis parameter.
	[説明]	センチネル引数は省略引数に一致する必要があります。

W0521551	[メッセージ]	No #pragma start_map_region is currently active: pragma ignored.
	[説明]	#pragma start_map_region は現在有効ではありません : pragma は無視されました。
W0521553	[メッセージ]	Nonstandard empty wide character literal treated as L'0'.
	[説明]	L'0' のような空のワイド文字リテラルは標準ではありません。
W0521561	[メッセージ]	Predefined meaning of "*symbol*" discarded.
	[説明]	すでに宣言された "*symbol*" の意味はなくなりました。
W0521564	[メッセージ]	enum qualified name is nonstandard.
	[説明]	enum 指定子は標準ではありません。
W0521565	[メッセージ]	Anonymous union qualifier is nonstandard.
	[説明]	無名共用体の指定子は標準ではありません。
W0521566	[メッセージ]	Anonymous union qualifier is ignored.
	[説明]	無名共用体の指定子は無視されました。
W0521570	[メッセージ]	Nonstandard specifier ignored.
	[説明]	標準でない指定子は無視されました。
W0521607	[メッセージ]	関数 " *関数名* " はすでに定義されています。#pragma text を指定できません。
	[説明]	関数 " *関数名* " はすでに定義されています。#pragma text を指定できません。指定を無視します。
W0521611	[メッセージ]	関数 " *関数名* " はすでに他の #pragma *文字列* が指定されています。" *文字列* " は指定できません。指定を無視します。
W0521632	[メッセージ]	" *文字列* " は認識できない情報行です。無視します。
W0521635	[メッセージ]	変数 " *変数名* " にはシンボル・ファイル中ですでにセクション " *セクション名* " が指定されています。後の指定を無視します。
W0521636	[メッセージ]	変数 " *変数名* " にはシンボル・ファイル中ですでにセクション " *セクション名* " が指定されています。#pragma による指定を無視します。
W0521644	[メッセージ]	宣言がファイル末尾で完了していません。
	[説明]	宣言終了のセミコロンがないままファイル末尾に達しました。
	[メッセージ]	【CC-RX】 Definition at end of file not followed by a semicolon or a declarator.
	[説明]	【CC-RX】 宣言終了のセミコロンがないままファイル末尾に達しました。
W0521649	[メッセージ]	" *マクロ名* " とその置換テキストの間には空白が必要です。
	[対処方法]	マクロ名とその置換テキストの間に空白を入れて区切ってください。
	[メッセージ]	【CC-RX】 White space is required between the macro name "*macro name*" and its replacement text
	[対処方法]	マクロ名とその置換テキストの間に空白を入れて区切ってください。
W0521651	[メッセージ]	文字列 はマルチコア・デバイスのみに指定できます。

W0523042	［メッセージ］	SuperH コンパイラとの互換性に影響ある " *機能項目* "(オプションや #pragma など) が使用されています。
	［対処方法］	SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
	［メッセージ］	【CC-RX】 Using *xxx* function at influence the code generation of "SuperH" compiler.
	［説明］	【CC-RX】 SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
W0523060	［メッセージ］	セクション指定が矛盾しています。
	［説明］	同一識別子が複数回宣言されており，宣言ごとに異なるセクションが指定されています。最初のセクションを有効にします。
	［メッセージ］	【CC-RX】 Incompatible section specified.
	［説明］	【CC-RX】 同一識別子が複数回宣言されており，宣言ごとに異なるセクションが指定されています。最初のセクションを有効にします。
W0523061	［メッセージ］	実引数は組み込み関数の仮引数と適合しません。
W0523062	［メッセージ］	返却値の型が組み込み関数の型と合っていません。
W0523063	［メッセージ］	" *文字列* " has no effect in this version.
	［説明］	このバージョンでは " *文字列* " は無効です。
W0523064	［メッセージ］	Address taken " *変数名* ". It may cause an upset endian indirect reference.
	［説明］	endian オプションと異なるエンディアンの 8 バイト変数 " *変数名* " のアドレスが取得されました。エンディアン処理が正しくない間接参照を引き起こす可能性があります。
W0523068	［メッセージ］	"*cpu*" でアトミック転送関数が使用されています。
W0523076	［メッセージ］	関数宣言はプロトタイプが必要です。
	［説明］	プロトタイプでない形式で関数宣言をしています。プロトタイプでない形式の関数宣言は，near ポインタを引数渡しする際に効率が悪くなる可能性があります。
W0523077	［メッセージ］	呼び出される関数はプロトタイプが必要です。
	［説明］	この関数呼び出しは，プロトタイプなしの関数型を使用しています。関数定義側がプロトタイプありの場合は，引数の受け渡しに不整合が出る可能性があります。 関数ポインタ経由で呼び出しの場合は，関数ポインタの型にプロトタイプが無いことを意味します。
W0530809	［メッセージ］	" *変数名* " で示す変数の const 修飾がファイル間で異なっています。
	［メッセージ］	【CC-RX】 const qualifier for variable " *変数名* " differs between files.
	［説明］	【CC-RX】 " *変数名* " で示す変数の const 修飾がファイル間で異なっています。
W0530811	［メッセージ］	" *シンボル名* " で示すシンボルの型がファイル間で異なっています。
	［メッセージ］	【CC-RX】 Type of symbol " *シンボル名* " differs between files.
	［説明］	【CC-RX】 " *シンボル名* " で示すシンボルの型がファイル間で異なっています。
W0533003	［メッセージ］	シフト数 (*数値*) が定められた範囲を越えています。
W0533004	［メッセージ］	比較結果が常に*文字列*です。
W0533005	［メッセージ］	0 で整数除算または整数剰余算を行いました。

W0542101	［メッセージ］	switch 文に対する分岐テーブルが多すぎます。-Xswitch=table オプションを無視します。
	［説明］	switch 文に対する分岐テーブルが多すぎます。ラベルの最大値と最小値の差が 8192 を越える switch 文に対しては，-Xswitch=table オプションを無視します。
W0544001	［メッセージ］	Alignment of " *セクション名* " sections is inconsistent. " *数値* " is assumed.
	［説明］	セクションの命名に誤りがあります。整列条件数の異なるセクションに同じ名前が付いています。
W0544002	［メッセージ］	Endian of " *セクション名* " sections is inconsistent. " *エンディアン種別* " is assumed.
	［説明］	セクションの命名に誤りがあります。エンディアンの異なるセクションに同じ名前が付いています。
W0550001	［メッセージ］	マクロ呼び出し時に指定された実引数が多すぎます。
	［対処方法］	実引数を確認してください。
W0550002	［メッセージ］	*option1* オプションと *option2* オプションを同時に指定することはできません。
	［説明］	*option1* オプションと *option2* オプションを同時に指定することはできません。 *option2* オプションは無視されます。
	［対処方法］	オプション指定を確認してください。
W0550003	［メッセージ］	*option* オプションには引数の指定が必要です。
	［説明］	*option* オプションには引数の指定が必要です。オプション指定は無視されます。
	［対処方法］	オプション指定の引数を確認してください。
W0550004	［メッセージ］	*option* オプションに指定された値が不正です。
	［説明］	*option* オプションに指定された値が不正です。オプション指定は無視されます。
	［対処方法］	オプション指定の値を確認してください。
W0550005	［メッセージ］	*option* オプションに指定されたシンボル *symbol* が不正です。
	［説明］	*option* オプションに指定されたシンボル *symbol* が不正です。オプション指定は無視されます。
	［対処方法］	オプション指定のシンボルを確認してください。
W0550006	［メッセージ］	*option* オプションに指定された引数が不正です。
	［説明］	*option* オプションに指定された引数が不正です。オプション指定は無視されます。
	［対処方法］	オプション指定の引数を確認してください。
W0550007	［メッセージ］	*core* コア共通のオブジェクト生成を指定する *option* オプションと、-C オプションで指定したデバイス・ファイルに不整合があります。
	［説明］	*core* コア共通のオブジェクト生成を指定する *option* オプションと，-C オプションで指定したデバイス・ファイルに不整合があります。 -C オプション指定を無視し，*core* コア共通オブジェクトを生成します。
	［対処方法］	オプション指定を確認してください。
W0550008	［メッセージ］	*option* オプションは *core* コアではサポートしていません。
	［説明］	*option* オプションは *core* コアではサポートしていません。オプション指定は無視されます。
	［対処方法］	オプション指定を確認してください。

W0550009	［メッセージ］	プログラマブル周辺 I/O レジスタの情報が存在しません。
	［説明］	プログラマブル周辺 I/O レジスタの情報が存在しません。-Xprogrammable_io オプションは無視されます。
	［対処方法］	オプション指定を確認してください。
W0550010	［メッセージ］	ディスプレースメントの値が指定可能な値の範囲を越えています。
	［説明］	ディスプレースメントの値が指定可能な値の範囲を越えています。 下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。
	［対処方法］	ディスプレースメントの値を確認してください。
W0550011	［メッセージ］	イミーディエトの値が指定可能な値の範囲を越えています。
	［説明］	イミーディエトの値が指定可能な値の範囲を越えています。 下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。
	［対処方法］	イミーディエトの値を確認してください。
W0550012	［メッセージ］	オペランドに指定した値が指定可能な値の範囲を越えています。
	［説明］	オペランドに指定した値が指定可能な値の範囲を越えています。 下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。
	［対処方法］	オペランドの値を確認してください。
W0550013	［メッセージ］	*register* が、*kind* レジスタとしてオペランドに指定されています。
	［対処方法］	レジスタの指定を確認してください。
W0550014	［メッセージ］	prepare/dispose 命令のレジスタ・リストに指定した値が不正です。
	［説明］	prepare/dispose 命令のレジスタ・リストに指定した値が不正です。 下位の有効な桁数分だけが指定されたものとみなし，アセンブルを続行します。
	［対処方法］	レジスタ・リストの値を確認してください。
W0550015	［メッセージ］	prepare/dispose 命令のレジスタ・リストに指定したレジスタが不正です。
	［説明］	prepare/dispose 命令のレジスタ・リストに指定したレジスタが不正です。 不正なレジスタを無視して，アセンブルを続行します。
	［対処方法］	レジスタ・リストのレジスタを確認してください。
W0550016	［メッセージ］	オペランドに、異なるアクセス幅の内部周辺 I/O レジスタを指定しています。
	［対処方法］	オペランドの内部周辺 I/O レジスタを確認してください。
W0550017	［メッセージ］	sld/sst 命令のベース・レジスタに ep 以外を指定しています。
	［対処方法］	ベース・レジスタの指定を確認してください。
W0550018	［メッセージ］	inst 命令に指定した番号のシステム・レジスタはアクセス禁止です。
	［対処方法］	システム・レジスタの番号を確認してください。
W0550019	［メッセージ］	オペランドに指定した値は 4 の倍数である必要があります。
	［説明］	オペランドに指定した値は 4 の倍数である必要があります。 端数を切捨てて，アセンブルを続行します。
	［対処方法］	オペランドの値を確認してください。

W0550020	［メッセージ］	-C オプションによる指定と $PROCESSOR 制御命令による指定が異なっています。
	［説明］	-C オプションによる指定と $PROCESSOR 制御命令による指定が異なっています。-C オプションを優先し，$PROCESSOR 制御命令によるターゲット・デバイス指定は無視されます。
	［対処方法］	オプションの指定を確認してください。
W0550021	［メッセージ］	*string* が、以前に指定したレジスタ数と異なる数で指定されています。
	［説明］	*string* が，以前に指定したレジスタ数と異なる数で指定されています。すでに指定されている数を使用します。この指定は無視されます。
	［対処方法］	レジスタ数を確認してください。
W0550022	［メッセージ］	*option* が、複数回指定されています。
	［説明］	*option* が，複数回指定されています。すでに指定されているオプションを使用します。この指定は無視されます。
	［対処方法］	オプション指定を確認してください。
W0550023	［メッセージ］	-Xprogrammable_io に指定された値が、デバイスの許容する範囲 (0x0,*value1*-*value2*) 外です。
	［説明］	-Xprogrammable_io に指定された値が，デバイスの許容する範囲 (0x0,*value1*-*value2*) 外です。 指定された値は無視して，デバイスの初期値を使用します。
	［対処方法］	オプションの値を確認してください。
W0550024	［メッセージ］	*option* オプションはインプリメントされていません。
	［説明］	*option* オプションはインプリメントされていません。無視されます。
	［対処方法］	オプションの指定を確認してください。
W0550026	［メッセージ］	奇数番号の付いたレジスタ（r*XX*）が指定されています。 偶数番号の付いたレジスタ（r*YY*）を指定したとして、アセンブルを続行します。
	［説明］	奇数番号の付いたレジスタ（r1，r3，…，r31）が指定されています。 指定できる汎用レジスタは，偶数番号の付いたレジスタ（r0，r2，r4，…，r30）だけです。 偶数番号の付いたレジスタ（r0，r2，r4，…，r30）を指定したとして，アセンブルを続行します。
	［対処方法］	レジスタの指定を確認してください。
W0550027	［メッセージ］	*control* 制御命令による指定が、以前の指定と異なっています。
	［説明］	*control* 制御命令による指定が，以前の指定と異なっています。以前の指定を優先し，後から指定した *control* 制御命令によるレジスタ・モード指定は無視されます。
	［対処方法］	*control* 制御命令の指定を確認してください。
W0550028	［メッセージ］	-Xreg_mode オプションによる指定と $REG_MODE 制御命令による指定が異なっています。
	［説明］	-Xreg_mode オプションによる指定と $REG_MODE 制御命令による指定が異なっています。-Xreg_mode オプションを優先し，$REG_MODE 制御命令によるレジスタ・モード指定は無視されます。
	［対処方法］	オプションの指定を確認してください。
W0550029	［メッセージ］	*デバイス名*指定時には，mul/mulu 命令のデスティネーション・オペランドに r0 を指定することはできません。
	［対処方法］	オペランドを確認してください。

W0550030	［メッセージ］	*デバイス名*指定時には，mul/mulu 命令のオペランドに X,Y,Y という形式を指定することはできません。
	［対処方法］	オペランドを確認してください。
W0550031	［メッセージ］	定義されていない識別子 *identifier* が参照されています。
	［対処方法］	識別子を確認してください。
W0550032	［メッセージ］	cache 命令が cll 命令として使用されています。
	［対処方法］	cache 命令を cll 命令として使用することは推奨されていません。cll 命令を記述してください。
W0550605	［メッセージ］	" *文字列* " オプションで指定された " パス名 " はファイルです。フォルダを指定してください。
W0550606	［メッセージ］	" *文字列* " オプションで指定されたフォルダ " フォルダ名 " が見つかりません。
W0550623	［メッセージ］	" *文字列 1*" オプションが指定されたので " *文字列 2*" オプションは無視しました。
W0550644	［メッセージ］	"-C" オプションと "-Xcommon" オプションが一致しません。" *文字列 1*" オプションの命令セットを 無視します。" *文字列 2*" 命令セットのコモン・オブジェクトを生成します。
W0550646	［メッセージ］	" *文字列 2*" オプションで指定された " *文字列 1*" は C 言語のマクロでは使用できません。アセンブラ のみに指定されたとみなします。
W0550647	［メッセージ］	" *文字列* " オプションが複数指定されています。後の指定が有効になります。
W0550649	［メッセージ］	" *文字列 1*" オプションと " *文字列 2*" オプションが矛盾しています。" *文字列 2*" オプションを無視します。
W0550651	［メッセージ］	" *文字列 1*" オプションが指定されていないので、" *文字列 2*" オプションを無視します。
W0550656	［メッセージ］	"-Xdev_path" オプションで指定されたフォルダ上に、デバイス・ファイルが見つかりません。
	［説明］	“-Xdev_path” オプションで指定されたフォルダ上に，デバイス・ファイルが見つかりません。標準のデバイス・ファイル・フォルダで探します。
W0551000	［メッセージ］	'.ALIGN' with not 'ALIGN' specified relocatable section.
	［説明］	ALIGN 指定がないセクション内に制御命令 ".ALIGN" が記述されています。
	［対処方法］	制御命令 ".ALIGN" の記述位置を確認してください。制御命令 ".ALIGN" を記述するセクションのセクション定義行に ALIGN 指定を記述してください。
W0551001	［メッセージ］	Destination address may be changed.
	［説明］	分岐先が期待するものと異なる位置になる可能性があります。
	［対処方法］	アドレッシングモードが最適選択されないように分岐命令のオペランドを記述してください。
W0551002	［メッセージ］	Floating point value is out of range.
	［説明］	浮動小数点数が範囲外です。
	［対処方法］	浮動小数点数の記述を確認してください。範囲外は無視します。
W0551003	［メッセージ］	Location counter exceed.
	［説明］	ロケーションカウンタが 0FFFFFFFFh を越えました。
	［対処方法］	.ORG のオペランド値を確認してください。ソースを記述し直してください。

W0551004	［メッセージ］	'.ALIGN' size is different.
	［説明］	整列条件値が異なります。
	［対処方法］	整列条件値を確認してください。
W0551006	［メッセージ］	Data in 'CODE' section align in 4byte.
	［説明］	endian=big 時，CODE セクション中データ領域の開始位置は 4 バイト境界に補正されます。
W0551007	［メッセージ］	Data size in 'CODE' section align in 4byte.
	［説明］	endian=big 時，CODE セクション中データ領域のサイズは 4 の倍数に補正されます。
W0551009	［メッセージ］	Multiple symbols.
	［説明］	.STACK でシンボルへのスタック値指定を重複して行なっています。
W0551010	［メッセージ］	Section attribute mismatch.
	［説明］	セクションの属性が異なっています。
W0551011	［メッセージ］	Use PM instruction.
	［説明］	特権命令を使用しています。
W0551012	［メッセージ］	Use FPU instruction.
	［説明］	浮動小数点演算命令を使用しています。
W0551013	［メッセージ］	Use DSP instruction.
	［説明］	DSP 機能命令を使用しています。
W0551014	［メッセージ］	Too many actual macro parameters.
	［説明］	マクロ実引数の数が多すぎます。余分な実引数は無視されます。
W0551015	［メッセージ］	Actual macro parameters are not enough.
	［説明］	マクロ実引数の数がマクロ仮引数の数より少なくなっています。該当する実引数のない仮引数は無効となります。
W0551016	［メッセージ］	'.END' statement is in include file.
	［説明］	インクルード・ファイルに .END 記述があります。.END を無視して処理します。
	［対処方法］	インクルード・ファイル内には，.END は記述できません。記述を削除してください。
W0550032	［メッセージ］	奇数番号の付いたベクトル・レジスタ（vrXX）が指定されています。
	［説明］	奇数番号の付いたベクトル・レジスタ（vr1，vr3，…，vr31）が指定されています。 指定できるベクトル・レジスタは，偶数番号の付いたベクトル・レジスタ（vr0，vr2，vr4，…，vr30）だけです。 偶数番号の付いたベクトル・レジスタ（vr0，vr2，vr4，…，vr30）を指定したとして，アセンブルを続行します。
	［対処方法］	ベクトル・レジスタの指定を確認してください。
W0560111	［メッセージ］	同じファイルが入力ファイルとして複数回指定されています。
W0560116	［メッセージ］	FPU 使用の不明なファイルが入力されています。
W0560205	［メッセージ］	シンボル “*symbol*” は、“*file*” において定義されている同名のシンボルと型が異なっています。

W0560206	[メッセージ]	シンボル “*symbol*” は、“*file*” において定義されている同名のシンボルとサイズが異なっています。
W0560207	[メッセージ]	シンボル “*symbol*” は、“*file*” において定義されている同名のシンボルと整列条件が異なっています。
W0560212	[メッセージ]	sbss または bss 属性セクションに割り付けられるシンボル *symbol* のサイズが 0 でした。
W0560214	[メッセージ]	$EXT_FUNC 制御命令で指定されたグローバル・シンボル “*symbol*” が定義されていません。
W0560305	[メッセージ]	セクション・タイプ (*section type*) を持つセクション名 “*section*” のセクションは、本リンカにおいてサポートしていません。無視しました。
W0560308	[メッセージ]	レジスタ・モードの異なるファイルが入力されています。-Xregmode_info オプションを指定すると、より詳細な情報を出力します。
W0560312	[メッセージ]	“*string*” オプションにより、“*section*” セクションを上書きします。
W0560313	[メッセージ]	フラッシュ・マスク・オプションの設定不可能機能を設定しました。
W0560314	[メッセージ]	内蔵命令 RAM に配置したセクション “*secton*” の整列条件は、4 の倍数を指定してください。
W0560315	[メッセージ]	内蔵 ROM/ 内蔵命令 RAM には、書き込み属性を持つセクション “*secton*” の配置は不適切です。
W0560316	[メッセージ]	レジスタ・モードの異なるファイルが入力されています。
W0560401	[メッセージ]	セクション “*section*” 内のリロケーション・エントリが不正なリロケーション・タイプ (*number*) を持っています。このエントリは無視されました。
W0560402	[メッセージ]	セクション “*section*” のオフセット (*offset*) に存在する参照をリロケートするための *number* 番目のシンボル・テーブル・エントリが見つかりません。このリロケーションは無視しました。
W0560403	[メッセージ]	セクション “*section2*” のリロケーションに用いられるリロケーション情報セクション “*section1*” 内のエントリが不正なリロケーション・オフセット (*offset*) を持っています。このエントリは無視しました。
W0560406	[メッセージ]	GP 相対リロケーションのための GP シンボルが存在しません (LOCAL) (ファイル : *file2*, セグメント :*segment*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。ローカル・シンボルは “*section1*”(ファイル :*file1*) に存在します。
W0560408	[メッセージ]	GP 相対リロケーションのための GP シンボルが存在しません (GLOBAL) 。 (ファイル : *file2*, セグメント :*segment*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。シンボルは “*section1*”(ファイル :*file1*) に存在します。
W0560410	[メッセージ]	ブート側実行ファイルのセクション “*section*” に GP 相対リロケーションのための GP シンボルが存在しません。 (ファイル : *file2*, セクション : *section2*, オフセット : *offset*, タイプ : *relocation type*) 。シンボル “*symbol*” は “*section1*”(ファイル :*file1*) に存在します。
W0560412	[メッセージ]	分岐系のリロケーション・エントリ（ファイル :*file*, セクション :*section*, オフセット :*offset*, リロケーション・タイプ :*relocation type*）によってリロケートされた値 (*value*) が奇数になっています。
W0560413	[メッセージ]	ロード／ストア系のリロケーション・エントリ（ファイル :*file*, セクション :*section*, オフセット :*offset*, リロケーション・タイプ :*relocation type*）によってリロケートされた値 (*value*) が奇数になっています。
W0560416	[メッセージ]	出力セクション *section2* から、出力セクション *section1* に配置されたシンボル “*symbol*” への分岐命令が、分岐可能範囲を越えています。分岐命令は (値 :*value*, ファイル :*file* 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。

W0560418	[メッセージ]	出力セクション *section2* から、出力セクション *section1* に配置されたローカル・シンボルへの分岐命令が、分岐可能範囲を越えています。分岐命令は (値 :*value*, ファイル :*file* 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。
W0560420	[メッセージ]	出力セクション *section1* がオーバフローしました。または、シンボル "*symbol*" の配置されたセクションとラベルの参照方法がファイル "*file*" において不正です。不正な参照は、(値 :*value*, 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。シンボル "*symbol*" はセクション "*section1*" に配置されました (ファイル :*file1*)。
W0560422	[メッセージ]	出力セクション *section1* がオーバフローしました。または、ローカル・シンボルの配置されたセクションとラベルの参照方法がファイル "*file2*" において不正です。不正な参照は、(値 :*value*, 入力セクション :*section3*, オフセット :*offset*, タイプ :*relocation type*) に存在します。ローカル・シンボルはセクション "*section1*" に配置されました (ファイル :*file1*)。
W0560501	[メッセージ]	ライブラリ・シンボル・テーブルが見つかりません。このライブラリ・ファイルを無視します。
W0560625	[メッセージ]	奇数の値（*number1*）を偶数の値（*number2*）に整列しました。
W0560628	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域がターゲット・マシンにおける可能なメモリ空間の範囲を越えています。
W0560639	[メッセージ]	-Xrelinkable_object オプションが指定された場合、セグメント・ディレクティブにおいて "*stirng*" を指定することはできません。無視しました。
W0560640	[メッセージ]	-Xrelinkable_object オプションが指定された場合、セクション・ディレクティブにおいて "stirng" を指定することはできません。無視しました。
W0560642	[メッセージ]	ファイル "*file*" 内のセクション "*section*" を割り付けることができるセグメント・タイプ LOAD を持つセグメントが存在しません。このセクションは、プログラム・ヘッダを持たないロード可能でないダミーのセグメントに割り付けられます。
W0560643	[メッセージ]	1 パスモードで "*stirng*" を指定することはできません。無視しました。
W0560644	[メッセージ]	-Xalign_fill オプション指定時に "*stirng*" を指定することはできません。"*stirng*" を無視しました。
W0560645	[メッセージ]	セグメント "*segment*" に対する *string* シンボルが多重に定義されています。最初に定義されたシンボル "*symbol*" を用います。
W0560647	[メッセージ]	*string* シンボルが多重に定義されています。最初に定義された "*symbol*" を用います。
W0560651	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域 (*number1-number2*) が *string* 相対で参照可能な範囲 (*number3-number4*) を越えています。
W0560653	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域 (*number1-number2*) がターゲット・マシンにおける可能な内蔵メモリ空間の範囲 (*number3-number4*) を越えています。
W0560654	[メッセージ]	EP シンボルディレクティブにおいて "*string*" を指定することはできません。無視しました。
W0560655	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域 (*number1-number2*) がガード (使用禁止) 領域 (*number3-number4*) にオーバラップしています。
W0560656	[メッセージ]	セグメント "*segment*" の割り付けが指示された領域 (*number1-number2*) がプログラマブル周辺 I/O 領域 (*number3-number4*) にオーバラップしています。
W0560659	[メッセージ]	セグメント "*segment*" のアドレス未指定時には指定できません。
W0560702	[メッセージ]	"*string2*" オプションが指定された場合、"*string1*" オプションを指定することはできません。 "*string1*" オプションは無視しました。

W0560706	[メッセージ]	"*stiring*" オプションに指定された奇数値 (*value1*) を偶数値 (*value2*) に整列しました。
W0560709	[メッセージ]	"*string1*" オプションは、"*stiring2*" コアのデバイスにおいてサポートしていません。
W0560710	[メッセージ]	新しい割り込みタイプを持つデバイス・ファイルが指定されたため、セグメントのアドレス順ソートを実行します。
W0560711	[メッセージ]	"*string*" オプションは指定したデバイスにおいてサポートしていません。オプション指定を無視しました。
W0561000	[メッセージ]	Option " *オプション* " ignored
	[説明]	" *オプション* " は無効です。" *オプション* " を無視します。
W0561001	[メッセージ]	Option " *オプション1* " is ineffective without option " *オプション2* "
	[説明]	" *オプション1* " は " *オプション2* " が必要です。" *オプション1* " を無視します。
W0561002	[メッセージ]	Option " *オプション1* " cannot be combined with option " *オプション2* "
	[説明]	" *オプション1* " と " *オプション2* " は同時に指定できません。" *オプション1* " を無視します。
W0561003	[メッセージ]	Divided output file cannot be combined with option " *オプション* "
	[説明]	" *オプション* " 指定時，出力ファイルの分割指定はできません。オプションの指定を無視します。先頭入力ファイル名を出力ファイル名として使用します。
W0561004	[メッセージ]	Fatal level message cannot be changed to other level : " *番号* "
	[説明]	アボート・エラー・メッセージはレベル変更できません。" *番号* " の指定を無視します。change_message オプションで変更できるエラーは，Information/Warning/Error レベルです。
W0561005	[メッセージ]	Subcommand file terminated with end option instead of exit option
	[説明]	end オプションの後に処理指定がありません。exit オプションを仮定して処理します。
W0561006	[メッセージ]	Options following exit option ignored
	[説明]	exit オプションの後のオプションを無視しました。
W0561007	[メッセージ]	Duplicate option : " *オプション* "
	[説明]	" *オプション* " が重複しています。最後に指定したオプションを有効にします。
W0561008	[メッセージ]	Option " *オプション* " is effective only in cpu type " *マイコン種別* "
	[説明]	" *オプション* " は " *マイコン種別* " 以外では無効です。" *オプション* " を無視します。
W0561010	[メッセージ]	Duplicate file specified in option " *オプション* " : " *ファイル名* "
	[説明]	" *オプション* " で同じファイルを二度指定しました。二度目の指定を無視します。
W0561011	[メッセージ]	Duplicate module specified in option " *オプション* " : " *モジュール* "
	[説明]	" *オプション* " で同じモジュールを二度指定しました。二度目の指定を無視します。
W0561012	[メッセージ]	Duplicate symbol/section specified in option " *オプション* " : " *名前* "
	[説明]	" *オプション* " で同じシンボル名またはセクション名を二度指定しました。二度目の指定を無視します。
W0561013	[メッセージ]	Duplicate number specified in option " *オプション* " : " *番号* "
	[説明]	" *オプション* " で同じエラー番号を指定しました。最後に指定した方を有効にします。

W0561017	[メッセージ]	セクション "*section*" はすでに -Xrompsec_data オプションにより指定されています。今回の指定を無視します。
W0561018	[メッセージ]	セクション "*section*" はすでに -Xrompsec_text オプションにより指定されています。今回の指定を無視します。
W0561100	[メッセージ]	Cannot find " *名前* " specified in option " *オプション* "
	[説明]	" *オプション* " で指定したシンボル名，またはセクション名が見つかりません。" *名前* " の指定を無視します。
W0561101	[メッセージ]	" *名前* " in rename option conflicts between symbol and section
	[説明]	rename オプションで指定した " *名前* " がセクション名とシンボル名の両方に存在します。シンボル名を変更の対象にします。
W0561102	[メッセージ]	Symbol " *シンボル* " redefined in option " *オプション* "
	[説明]	" *オプション* " で指定したシンボルはすでに定義されています。そのまま処理を続けます。
W0561103	[メッセージ]	Invalid address value specified in option " *オプション* " : " *アドレス* "
	[説明]	" *オプション* " で指定した " *アドレス* " は無効な値です。" *アドレス* " の指定を無視します。
W0561104	[メッセージ]	Invalid section specified in option " *オプション* " : " *セクション* "
	[説明]	" *オプション* " で無効なセクションを指定しています。
	[対処方法]	以下を確認してください。 (1) -output オプションは，初期値のないセクションを指定できません。 (2) -jump_entries_for_pic オプションは，プログラムセクション以外を指定できません。
W0561110	[メッセージ]	Entry symbol " *シンボル* " in entry option conflicts
	[説明]	entry オプションで指定した " *シンボル* " 以外のシンボルがコンパイル，アセンブル時にエントリ・シンボルとして指定されています。オプション指定を優先します。
W0561120	[メッセージ]	Section address is not assigned to " *セクション* "
	[説明]	" *セクション* " のアドレス指定がありません。" *セクション* " を最後尾に配置します。
	[対処方法]	rlink オプション -start を使用して，セクションのアドレスを設定してください。
W0561121	[メッセージ]	Address cannot be assigned to absolute section " *セクション* " in start option
	[説明]	" *セクション* " は絶対アドレス・セクションです。絶対アドレス・セクションに対するアドレス指定を無視します。
W0561122	[メッセージ]	Section address in start option is incompatible with alignment : " *セクション* "
	[説明]	start オプションで指定した " *セクション* " のアドレスは整列条件数と矛盾しています。整列条件数に合わせてセクションアドレスを補正します。
W0561130	[メッセージ]	Section attribute mismatch in rom option : " *セクション1*, *セクション2* "
	[説明]	rom オプションで指定した " *セクション1* " と " *セクション2* " の属性，整列条件数が異なります。" *セクション2* " の整列条件数はどちらか大きい方を有効とします。
W0561140	[メッセージ]	Load address overflowed out of record-type in option " *オプション* "
	[説明]	アドレス値よりも小さい record 形式を指定しました。指定した record 形式を越える範囲は，別の record 形式で出力します。

W0561141	[メッセージ]	Cannot fill unused area from " *アドレス* " with the specified value
	[説明]	空きエリアのサイズが space オプションで指定された値の倍数となっていないため，" *アドレス* " 以降に指定データを出力できませんでした。
W0561150	[メッセージ]	Sections in " *オプション* " option have no symbol
	[説明]	" *オプション* " で指定したセクションは外部定義シンボルがありません。
W0561160	[メッセージ]	Undefined external symbol " *シンボル* "
	[説明]	未定義の " *シンボル* " を参照しています。
W0561180	[メッセージ]	Directive command " *制御命令* " is duplicated in " *ファイル* "
	[説明]	複数のソース・ファイルに，" *制御命令* " を記述しています。 " *制御命令* " は，複数記述することはできません。
W0561181	[メッセージ]	Fail to write " *出力コード種別* "
	[説明]	出力ファイルへの，" *出力コード種別* " の書き込みを失敗しました。 出力ファイルに，" *出力コード種別* " の書き込み先アドレスが含まれていない可能性があります。 出力コード種別： CRC コード書き込み失敗時： "CRC Code"
W0561182	[メッセージ]	Cannot generate vector table section " *セクション* "
	[説明]	入力ファイル内に，ベクタテーブル " *セクション* " があります。最適化リンカは，" *セクション* " を自動生成しません。
W0561183	[メッセージ]	Interrupt number " *ベクタ番号* " of " *セクション* " is defined in input file
	[説明]	VECTN オプションで記述したベクタ番号は，入力ファイル内で定義済みです。入力ファイルの内容を優先して，処理を継続します。
W0561190	[メッセージ]	Section " *セクション* " was moved other area specified in option "*cpu*=< *メモリ属性* >"
	[説明]	外部変数アクセス最適化によりオブジェクト・サイズが変更されたため，次の cpu 指定範囲の " *セクション* " を移動しました。
W0561191	[メッセージ]	Area of "FIX" is within the range of the area specified by "*cpu*=< *メモリ属性* >" :"<*start*>-<*end*>"
	[説明]	cpu オプションで，メモリ属性 FIX と FIX 以外の <*start*>-<*end*> 範囲が重複していたため，FIX を有効にしました。
W0561192	[メッセージ]	Bss Section " *セクション名* " is not initialized
	[説明]	初期値なしのデータ・セクション " *セクション名* " は，初期設定プログラムで初期化できません。
	[対処方法]	-cpu 指定範囲，ポインタ変数のサイズ指定を見直してください。
W0561193	[メッセージ]	Section " *セクション名* " specified in option " *オプション* " is ignored
	[説明]	-cpu=stride の機能で分割したセクションの，後半部への " *オプション* " 指定は無効となります。
	[対処方法]	後半部のセクションは " *オプション* " で指定しないでください。

W0561194	［メッセージ］	Section " *セクション*" in relocation " *ファイル*"-" *セクション*"-" *オフセット*" is changed
	［説明］	" *セクション*"" *ファイル*"" *オフセット*" の位置にある " *セクション*" を参照していたリロケーションが，分割した後半セクションを参照するよう変更しました。
	［対処方法］	分割しないようにするには，" *セクション*" を contiguous_section オプションで指定してください。
W0561195	［メッセージ］	Read only data memory domains differ in " *ファイル*"
	［説明］	メモリ・モデルの異なる " *ファイル*" を入力しました。
	［対処方法］	コンパイラのオプションが正しいか確認してください。
W0561200	［メッセージ］	Backed up file " *ファイル1*" into " *ファイル2*"
	［説明］	入力ファイル " *ファイル1*" は書き換えられました。書き換える前の " *ファイル1*" の内容は " *ファイル2*" にバックアップされています。
W0561300	［メッセージ］	Option " *オプション*" is ineffective without debug information
	［説明］	入力ファイル内にデバッグ情報がありません。" *オプション*" 指定を無視します。
	［対処方法］	コンパイル，アセンブル時に該当するオプションを指定しているか確認してください。
W0561301	［メッセージ］	No inter-module optimization information in input files
	［説明］	入力ファイル内にモジュール間最適化情報がありません。optimize オプションを無視します。
	［対処方法］	コンパイル，アセンブル時に goptimize オプションを指定してください。
W0561302	［メッセージ］	No stack information in input files
	［説明］	入力ファイル内にスタック情報がありません。stack オプションを無視します。入力ファイルがアセンブラ出力ファイルの場合は，stack オプションは無効です。
W0561305	［メッセージ］	Entry address in " *ファイル*" conflicts : " *アドレス*"
	［説明］	異なるエントリーアドレスのファイルが複数入力されています。
W0561310	［メッセージ］	" *セクション*" in " *ファイル*" is not supported in this tool
	［説明］	" *ファイル*" 内に非サポートセクションがありました。" *セクション*" を無視します。
W0561311	［メッセージ］	Invalid debug information format in " *ファイル*"
	［説明］	" *ファイル*" 内のデバッグ情報は dwarf2 ではありません。debug 情報を削除します。
W0561320	［メッセージ］	Duplicate symbol " *シンボル*" in " *ファイル*"
	［説明］	" *シンボル*" は重複しています。先に入力したファイル内シンボルを優先します。
W0561321	［メッセージ］	Entry symbol " *シンボル*" in " *ファイル*" conflicts
	［説明］	エントリ・シンボル定義のあるオブジェクト・ファイルを複数入力しました。先に入力したファイル内のエントリ・シンボルを有効にします。
W0561322	［メッセージ］	Section alignment mismatch : " *セクション*"
	［説明］	整列条件数の異なる同名セクションを入力しました。整列条件数は最大の指定を有効にします。

W0561323	［メッセージ］	Section attribute mismatch : " *セクション* "
	［説明］	属性の異なる同名セクションを入力しました。絶対セクションと相対セクションの場合は，絶対セクションとして扱います。read/write 属性が異なる場合は，どちらも許可します。
W0561324	［メッセージ］	Symbol size mismatch : " *シンボル* " in " *ファイル* "
	［説明］	サイズの異なるコモン・シンボルまたは定義シンボルが入力されました。定義シンボルを優先します。コモン・シンボル同士の場合は，先に入力したファイル内シンボルを優先します。
W0561325	［メッセージ］	Symbol attribute mismatch : " *シンボル* ":" *ファイル* "
	［説明］	" *ファイル* " 内の " *シンボル* " が，ほかのファイルの同名シンボルと属性が一致していません。
	［対処方法］	シンボルを確認してください。
W0561326	［メッセージ］	Reserved symbol " *シンボル* "is defined in " *ファイル* "
	［説明］	予約された名称のシンボル " *シンボル* " が " *ファイル* " 内で定義されています。
W0561327	［メッセージ］	Section alignment in option "aligned_section" is small : " *セクション* "
	［説明］	aligned_section オプション指定時の整列条件数 16 の方が，" *セクション* " の整列条件数より小さいため，指定セクションに対するオプション指定を無視します。
W0561402	［メッセージ］	Parentheses specified in option "start" with optimization
	［説明］	start オプションで括弧 "()" を記述した場合，最適化機能は使用できません。 最適化機能を無効にします。
W0561410	［メッセージ］	Cannot optimize " *ファイル* "-" *セクション* " due to multi label relocation operation
	［説明］	複数ラベルのリロケーション演算を持つセクションは最適化できません。" *ファイル* " 内の " *セクション* " を最適化対象外にします。
W0561430	［メッセージ］	Cannot generate effective bls file for compiler optimization
	［説明］	無効な bls ファイルが生成されました。コンパイル時に，外部変数アクセス最適化 (map オプション) を指定しても，この最適化は実施できません。
	［対処方法］	コンパイラの外部変数アクセス最適化 (map オプション) には，以下の制限があります。該当する内容がないかを確認し，セクション配置を見直してください。 コンパイル時に base オプションを使用している場合，プログラムセクションの直後にデータ・セクションを配置すると，外部変数アクセス最適化が実施できない場合があります。 なお，bls ファイルは " 外部シンボル割り付け情報ファイル " を指します。これは，コンパイラの map オプションに使用するための情報ファイルです。
W0561500	［メッセージ］	Cannot check stack size
	［説明］	スタックセクションがないため，コンパイル時の stack オプションで指定したスタックサイズの整合性をチェックできません。
	［対処方法］	コンパイル時の stack オプションの整合性をチェックするためには，コンパイル時，アセンブル時に goptimize オプション指定が必要です。
W0561501	［メッセージ］	Stack size overflow : " *スタック・サイズ* "
	［説明］	スタック・セクション・サイズが，コンパイル時に stack オプションで指定した " *スタックサイズ* " を越えました。
	［対処方法］	コンパイル時のオプションを変更するか，スタック量を削減できるようにプログラムを変更してください。

W0561502	［メッセージ］	Stack size in " *ファイル* " conflicts with that in another file
	［説明］	複数のファイルで異なるスタックサイズを指定されています。
	［対処方法］	コンパイル時のオプションを確認してください。
W0561510	［メッセージ］	Input file was compiled with option "smap" and option "map" is specified at linkage
	［説明］	smap を指定してコンパイルしたファイルがあります。
	［対処方法］	smap を指定したファイルは，2 回目のビルドで map オプションを指定してコンパイルしないでください。
W0561520	［メッセージ］	" *ユーザ・オプション・バイト / オンチップ・デバッグ制御値* " in " *セクション* " created by device file
	［説明］	ユーザ・オプション・バイト／オンチップ・デバッグ制御値は，デバイス・ファイルの初期値を設定しました。
W0561521	［メッセージ］	Cannot generate section " *セクション* "
	［説明］	セクションを生成できません。
	［対処方法］	コンパイル時のオプションとソース・ファイルのセクション指定を確認してください。
W0561531	［メッセージ］	" *オプション* " ignored because Device file with " *タグ* " is required
	［説明］	" *オプション* " は無効です。
	［対処方法］	デバイス・ファイルを確認してください。
W0562010	［メッセージ］	-Xhex_symtab=global オプションと -Xhex_symtab=all オプションは、拡張テック・ヘキサ・フォーマットが指定された場合にのみ有効です。
W0562011	［メッセージ］	-Xhex_symtab=global オプションは、拡張テック・ヘキサ・フォーマットが指定されたが指定された場合にのみ有効です。
W0562012	［メッセージ］	-Xhex_symtab=all オプションは、-Xhex_symtab=global オプションと -Xhex_format=T オプションと同時に指定された場合にのみ有効です。
W0562013	［メッセージ］	“*string1*” オプションが指定されたため、“*string2*” オプションは無効となります。
W0562015	［メッセージ］	-Xhex_fill 指定時はエラー “*section*” セクションは、-Xhex_fill オプションで指定した領域の下限を越えています。 -Xhex_fill 指定時されてなければワーニング “*section*” セクションは、内蔵 ROM 領域の下限を越えています。
W0562016	［メッセージ］	“*section*” セクションは、内蔵メモリ空間の範囲を越えています。
W0562017	［メッセージ］	“*section*” セクションは、プログラムメモリの下限を越えています。
W0562018	［メッセージ］	“*section*” セクションの途中の指定されたアドレスからヘキサ変換を行います。
W0562019	［メッセージ］	“*section*” セクションの途中の指定された領域までヘキサ変換を行います。

W0562020	［メッセージ］	アドレスが、インテル拡張ヘキサ・フォーマット形式で表現可能なアドレスの最大値（20 ビット）を越えています。表現可能な範囲をアドレスとして出力する形で処理を続行します。
	［説明］	本メッセージの原因としては，次の理由が考えられます。 (1) ROM 化を忘れている ROM に配置したセクションと同時に内蔵 RAM のセクションをヘキサ変換を行おうとしている可能性があります。ヘキサ変換を行う前に ROM 化を行ってください（ROM 化についての詳細は，「8.5 ROM 化」を参照してください）。 (2) ヘキサ変換対象のセクションが間違っている 同時にヘキサ変換するセクションのアドレスが，大きく離れている可能性があります。複数の ROM を使用している場合には，ROM ごとにヘキサ変換を行ってください。また，ヘキサ変換するセクションは正しいか，セクションの配置アドレスが正しいかを確認してください。 (3) セクションのサイズ自体が大きい ヘキサ変換を行おうとしている領域が大きく表現可能な範囲を越えている場合には，ヘキサ変換できません。表現可能な範囲まで分割するか，別のヘキサ・フォーマットを使用してください。 ［注意事項］ アドレスの最大値を 20 ビットと表現していますが，実際には 20 ビット付近となります。本メッセージが出力される原因であるインテル拡張ヘキサ・フォーマット形式の 20 ビットのアドレス表現は，上位アドレスを保有する拡張アドレス・レコードと，そこからのオフセットを所有するデータ・レコードから算出されます。拡張アドレス・レコードの上位アドレスが 20 ビットに収まっていれば，それに続くデータ・レコードのオフセットを加算した結果が 20 ビットを越えたとしても，インテル拡張ヘキサ・フォーマット形式として正常な出力であり，エラーにはなりません。
W0562021	［メッセージ］	アドレスがモトローラ S タイプ・ヘキサ・フォーマット（スタンダード・アドレス）形式で表現可能なアドレスの最大値（24 ビット）を越えています。 表現可能な範囲をアドレスとして出力する形で処理を続行します。
	［説明］	本メッセージの原因としては，次の理由が考えられます。 (1) ROM 化を忘れている ROM に配置したセクションと同時に内蔵 RAM のセクションをヘキサ変換を行おうとしている可能性があります。ヘキサ変換を行う前に ROM 化を行ってください（ROM 化についての詳細は，「8.5 ROM 化」を参照してください）。 (2) ヘキサ変換対象のセクションが間違っている 同時にヘキサ変換するセクションのアドレスが，大きく離れている可能性があります。複数の ROM を使用している場合には，ROM ごとにヘキサ変換を行ってください。また，ヘキサ変換するセクションは正しいか，セクションの配置アドレスが正しいかを確認してください。 (3) セクションのサイズ自体が大きい ヘキサ変換を行おうとしている領域が大きく表現可能な範囲を越えている場合には，ヘキサ変換できません。表現可能な範囲まで分割するか，別のヘキサ・フォーマットを使用してください。

W0562022	[メッセージ]	先頭アドレスが、インテル拡張ヘキサ・フォーマット形式で表現可能なアドレスの最大値（20 ビット）を越えています。表現可能な範囲をアドレスとして出力する形で処理を続行します。
	[説明]	本メッセージの原因としては，次の理由が考えられます。 (1) セクションのアドレスが大きい セクションの配置されているアドレスが，表現可能な範囲を越えている可能性があります。-Xhex_offset オプションでヘキサ変換する領域の先頭アドレスを指定してそのアドレスからのオフセットとしてください。 (2) -Xhex_offset オプションで指定した値が不適切 -Xhex_offset オプションを指定することで，アドレスを指定した値からのオフセットとして扱うことが可能です。 この値からのオフセットが表現可能な範囲を越えている可能性があります。適切な値を指定してください。
W0562023	[メッセージ]	先頭アドレスが、モトローラ S タイプ・ヘキサ・フォーマット形式で表現可能なアドレスの最大値（24 ビット）を越えています。表現可能な範囲をアドレスとして出力する形で処理を続行します。
	[説明]	本メッセージの原因としては，次の理由が考えられます。 (1) セクションのアドレスが大きい セクションの配置されているアドレスが，表現可能な範囲を越えている可能性があります。-Xhex_offset オプションでヘキサ変換する領域の先頭アドレスを指定してそのアドレスからのオフセットとしてください。 (2) -Xhex_offset オプションで指定した値が不適切 -Xhex_offset オプションを指定することで，アドレスを指定した値からのオフセットとして扱うことが可能です。 この値からのオフセットが表現可能な範囲を越えている可能性があります。適切な値を指定してください。
W0562024	[メッセージ]	指定されたブロック長の最大値が小さすぎます。デフォルトの値 *length* に変更し処理を続行します。
W0562025	[メッセージ]	指定されたブロック長の最大値が大きすぎます。指定することのできる値の最大値 *length* に変更し処理を続行します。
W0562026	[メッセージ]	ブロック長の最大値をデフォルトの値から *length* に変更し処理を続行します -Xhex_block_size オプション時に指定することのできる値が指定された場合に出力されます。
W0562027	[メッセージ]	シンボル・ブロックのブロック長が指定されたブロック長の最大値を越えています。
W0571600	[メッセージ]	An error occurred during name decoding of " *インスタンス* "
	[説明]	" *インスタンス* " がデコードできませんでした。エンコード名でメッセージ出力します。
W0578306	[メッセージ]	can not open file *file*
	[説明]	ファイル *file* をオープンできません。
W0578307	[メッセージ]	can not close file *file*
	[説明]	ファイル *file* をクローズできません。
W0578308	[メッセージ]	can not read file *file*
	[説明]	ファイル *file* からの読み込みができません。
W0578309	[メッセージ]	can not write file *file*
	[説明]	ファイル *file* への書き込みができません。

W0578310	[メッセージ]	can not seek file *file*
	[説明]	ファイル *file* をシークできません。
W0578311	[メッセージ]	can not find file *file*
	[説明]	ファイル *file* を見つけることができません。
W0578315	[メッセージ]	can not find member *member*
	[説明]	ライブラリ・ファイル内にメンバ *member* が存在しません。
W0578322	[メッセージ]	this symbol offset not true
	[説明]	ライブラリ・ファイル内のシンボルオフセットが不正です。
W0592009	[メッセージ]	入力ファイル "*file*" のオープンに失敗しました。
	[説明]	リスト・ファイルで，次のファイルが指定されていれば次のファイルの変換に移ります。
W0592011	[メッセージ]	"-l" オプションが指定されたので、入力ファイルの指定は無視されます。
	[説明]	リスト・ファイルと入力ファイルは同時に指定できません。"-l" オプションで指定されたリスト・ファイルの変換が実行され，入力ファイルは無視されます。
W0592012	[メッセージ]	"-c" の指定が、リスト・ファイルとコマンド・ラインで異なります。コマンド・ラインの指定は無視されます (*file*)。
	[説明]	リスト・ファイル中で指定された入力ファイル "*file*" に対応する "-c" オプションの指定がリスト・ファイルとコマンド・ラインで異なります。リスト・ファイルの指定で変換します。
W0592017	[メッセージ]	入力ファイル "*file*" が見つかりません。
	[説明]	入力ファイル "*file*" が見つかりませんでした。リスト・ファイルで，次のファイルが指定されていれば次のファイルの変換に移ります。
W0592103	[メッセージ]	認識できない #pragma '*string*' がありました。
	[説明]	変換前の C 言語ソース・ファイルで許可されていない #pragma があるため変換できません。
	[対処方法]	C 言語ソース・ファイルを編集して，構文エラーを修正してください。
W0592104	[メッセージ]	マクロ定義内の __interrupt/__multi_interrupt は変換できません。
	[説明]	マクロ定義内に書かれた __interrupt および __multi_interrupt は変換ができません。
	[対処方法]	手作業にて移植を行ってください。
W0592105	[メッセージ]	C ソース内に _rcopy() が記述されています。アセンブラ・ソースからの呼び出しに変更してください。
	[説明]	C ソース・プログラムから _rcopy 関数を参照しています。
	[対処方法]	アセンブラ・ソースからの呼び出しに変更するか，または第 1 引数として指定する変数を const 修飾子をつけて宣言してください。
W0594100	[メッセージ]	ファイル (*ファイル名*) が見つかりませんでした。ファイル名を指定し直してください。
	[説明]	指定されたファイルが存在しません。
	[対処方法]	ファイル名を指定し直してください。

W0594101	［メッセージ］	ファイル (*ファイル名*) は読み込み禁止です。 読み込める状態にしてください。
	［説明］	指定されたファイルは読み込み禁止です。
	［対処方法］	ファイルの属性をチェックして，読み込み可能な状態にしてください。
W0594102	［メッセージ］	ファイル (*ファイル名*) は書き込み禁止です。 ファイルおよびフォルダを書き込める状態にしてくだい。
	［説明］	指定されたファイルは書き込み禁止です。
	［対処方法］	ファイルおよびフォルダの属性をチェックして，書き込み可能な状態にしてください。
W0594103	［メッセージ］	ファイル (*ファイル名*) の読み込みでエラーが発生しました。ファイルが読み込み可能な状態にあるか確認してください。
	［説明］	指定されたファイルの読み込みでエラーが発生しました。
	［対処方法］	ファイルが読み込み可能な状態にあるか確認してください。
W0594104	［メッセージ］	ファイル (*ファイル名*) の書き込みでエラーが発生しました。ファイルが書き込み可能な状態にあるか確認してください。
	［説明］	指定されたファイルの書き込みでエラーが発生しました。
	［対処方法］	ファイルが書き込み可能な状態にあるか確認してください。
W0594105	［メッセージ］	ファイル (*ファイル名*) の *行番号*行は一行の文字数限界（*限界値*）を越えています。
W0594106	［メッセージ］	ファイル (*ファイル名*) の行数はファイルの行数限界（*限界値*）を越えています。
W0594107	［メッセージ］	ファイル名（*ファイル名*）が長すぎます。255 文字以上となるファイルを扱うことはできません。
W0594110	［メッセージ］	使われていない関数名 (*関数名*) が指定されています（*ファイル名* : *行番号*行）。 C ソース・ファイル中に定義または直接呼び出しがある関数を指定してください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，プロジェクトで使われていない関数名が指定されています。 指定できるのは，C ソース・ファイル中に定義，または直接呼び出しがある関数のみです。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。

W0594111	［メッセージ］	スタティック関数名の前に付けられたファイル名（*ファイル名*）が長すぎます （*ファイル名*：*行番号*行）。 255 文字以下にしてください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，スタティック関数の前に付けられた，その関数の定義を含むファイルの名前が長すぎるものが見つかりました。 255 文字以下にしてください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594112	［メッセージ］	関数名（*関数名*）が長すぎます（*ファイル名*：*行番号*行）。 1,022 文字以下にしてください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，長すぎる関数名が見つかりました。 1,022 文字以下にしてください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594113	［メッセージ］	“ADD=” の後の値（*値*）が不正です（*ファイル名*：*行番号*行）。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。 加算サイズは，“ADD=” の後に，10 進数，または “0x” か “0X” で始まる 16 進数で，指定可能な値を指定してください。 指定可能な値は次のとおりです。 0 ～ 2,147,483,647
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594114	［メッセージ］	複数の “ADD=” が指定されています（*ファイル名*：*行番号*行）。 加算サイズは一関数に 1 つしか指定できません。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な加算サイズの指定が見つかりました。 加算サイズは一関数に 1 つしか指定できません。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。

W0594115	［メッセージ］	“RECTIME=” の後の値（*値*）が不正です（*ファイル名*：*行番号*行）。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な再帰回数の指定が見つかりました。 再帰回数は，“RECTIME=” の後に，10 進数，または “0x” か “0X” で始まる 16 進数で正の値を指定してください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594116	［メッセージ］	複数の “RECTIME=” が指定されています（*ファイル名*：*行番号*行）。 再帰回数は一関数に 1 つしか指定できません。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な再帰回数の指定が見つかりました。 再帰回数は一関数に 1 つしか指定できません。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594117	［メッセージ］	再帰関数以外で “RECTIME=” が指定されています（*ファイル名*：*行番号*行）。 “RECTIME=” は再帰関数にしか使うことができません。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な再帰回数の指定が見つかりました。 再帰回数の指定 “RECTIME=” は再帰関数にしか使うことができません。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594118	［メッセージ］	“CALL=” の後に関数名が指定されていません（*ファイル名*：*行番号*行）。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な呼び出し関数の指定が見つかりました。 “CALL=” の後に関数名を指定してください。プロジェクトで使用されていない関数を指定することはできません。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。

W0594119	［メッセージ］	“CALL=” の後に複数の関数名が指定されています（*ファイル名*：*行番号*行）。 関数ごとに “CALL=” を用いて指定してください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，不正な呼び出し関数の指定が見つかりました。 “CALL=” の後には 1 つの関数しか指定できません。複数を指定する場合，”CALL ＝ ” も複数指定してください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594124	［メッセージ］	同一関数に対する指定が複数見つかりました（*ファイル名*：*行番号 1* 行と *行番号 2* 行）。 片方を削除してください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルで，同一関数に対する指定が *行番号 1* 行と *行番号 2* 行で見つかりました。 片方を削除してください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594125	［メッセージ］	ファイル (*ファイル名*) の *行番号*行は一行の文字数限界（5,119）を越えています。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルの *行番号*行は一行の文字数限界（5,119）を越えています。 呼び出し関数の指定を減らしてください。呼び出す関数すべてを登録しても，スタック・サイズに加算されるのは，スタック・サイズが最大となる関数だけです。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。
W0594126	［メッセージ］	不正なフォーマットが見つかりました（*ファイル名*：*行番号*行）。 スタックサイズ指定ファイルを確認してください。 読み込みを中止しますか？
	［説明］	スタック・サイズ指定ファイルの *行番号*行は不正なフォーマットです。 ファイルを確認してください。
	［対処方法］	［中止］： 処理を中止します。 ［再試行］： エラー行を無視して，次の行から読み込みを継続します。 ［無視］： エラー行を無視して，次の行から読み込みを継続します。以降の同様の警告は，メッセージ表示部には出力しますが，本ダイアログは表示しません。

W0594127	［メッセージ］	標準ライブラリ用のスタックサイズ指定ファイル (*ファイル名*) が見つかりませんでした。*ファイル名*が持っているサイズ情報 (*デフォルト・ファイル名*に同じ) を使います。
	［説明］	標準ライブラリ用のスタック・サイズ指定ファイルが見つかりませんでした。 skcx が持っているサイズ情報（*デフォルト・ファイル名*に同じ) を使います。
	［対処方法］	正常にインストールされていない可能性があるので，再インストールしてください。
W0594128	［メッセージ］	標準ライブラリ用のスタックサイズ指定ファイル（*ファイル名*）で不正なフォーマットが見つかりました（*行番号*行）。
	［説明］	標準ライブラリ用のスタック・サイズ指定ファイルで不正なフォーマットが見つかりました。
	［対処方法］	正常にインストールされていない可能性があるので，再インストールしてください。
W0594129	［メッセージ］	stk システムファイル (*ファイル名*) が壊れています。 無視しました。
	［説明］	stk システム・ファイルが不正です。
	［対処方法］	これより前に行ったスタック・サイズの変更が正しく反映されていない可能性があるので，確認のうえ，正しくない場合には再設定してください。
W0594130	［メッセージ］	中間アセンブラ・ファイルが見つかりませんでした。 CS+ で C ソース・ファイルを登録し、ビルド・ツールのプロパティ パネルで [コンパイル・オプション] タブを選択し、[出力ファイル] カテゴリの [アセンブラ・ソース・ファイルを出力する] プロパティで [はい (-Xasm_path)] を指定したうえで、リビルドしてください。
	［説明］	中間アセンブラ・ファイルが見つかりませんでした。
	［対処方法］	CS+ で C ソース・ファイルを登録し，ビルド・ツールのプロパティ パネルで［コンパイル・オプション］タブを選択し，［出力ファイル］カテゴリの［アセンブリ・ファイルを出力する］プロパティで［はい (-Xasm_path)］を指定したうえで，リビルドしてください。
W0594131	［メッセージ］	中間アセンブラ・ファイル (*ファイル名*) が見つかりませんでした。 対応する C ソース・ファイルのコンパイラオプションの設定を確認したうえで、リビルドしてください
	［説明］	中間アセンブラ・ファイルが見つかりません。
	［対処方法］	CS+ で，対応する C ソース・ファイルのプロパティ パネルで［個別コンパイラ・オプション］タブを選択し，［出力ファイル］カテゴリの［アセンブリ・ファイルを出力する］プロパティで［はい (-Fs)］を指定したうえで，リビルドしてください。
W0594132	［メッセージ］	不正なフォーマットが中間アセンブラ・ファイル（*ファイル名*）で見つかりました（*行番号*行）。 ファイルを確認してください。
	［説明］	中間アセンブラ・ファイルの*行番号*行は，不正なフォーマットです。
	［対処方法］	CX が生成した中間アセンブラ・ファイルでない可能性があります。ファイルを確認して，リビルドしてください
W0594133	［メッセージ］	*コンパイラ 名 バージョン*には対応していません。 *対応バージョン*以降を使用してください。
	［説明］	コンパイラのバージョンが古いため，ランタイムライブラリ，または標準ライブラリの呼び出しが含まれる場合には，正しいスタック・サイズを計算できません。
	［対処方法］	CX には対応しています。リビルドしてください。

W0594134	［メッセージ］	デバイスが異なるアセンブラ・ソース・モジュール・ファイル（*ファイル名*）が見つかりました。
	［説明］	ファイルは異なるシリーズ用のコンパイラで生成されたものです。 このファイルは解析できません。
W0594135	［メッセージ］	コンパイラのバージョンが異なるアセンブラ・ソース・モジュール・ファイル（*ファイル名*）が見つかりました。
	［説明］	アセンブラ・ソース・モジュール・ファイルは異なるバージョンのコンパイラでコンパイルされました。 このファイルは解析できません。
W0594140	［メッセージ］	プロジェクト・ファイル (*ファイル名*) が不正です。ファイルを確認してください。
	［説明］	プロジェクト・ファイルが正しくありません。ファイルを確認してください。 起動後にメニューから開いたプロジェクト・ファイルが不正な場合に表示されます。
W0594150	［メッセージ］	出力結果が 32,767 行を越えています。関数の選択を変更するか、最大経路のみの出力にしてください。
	［説明］	出力結果が 32,767 行を越えています。
	［対処方法］	関数の選択を変更するか，最大経路のみの出力にしてください。
W0594151	［メッセージ］	出力結果の一行が 5,119 文字を越えています。 関数の選択を変更するか、テキスト形式で出力してください。
	［説明］	出力結果の一行が 5,119 文字を越えています。 関数の選択を変更するか，テキスト形式で出力してください。テキスト形式の場合，改行して表示されます。
W0594160	［メッセージ］	呼び出し関数の数が多すぎます。 1,024 個以下にしてください。
	［説明］	呼び出し関数の指定数が限界値を越えています。 1,024 個以下にしてください。呼び出す関数をすべて登録しても，スタック・サイズに加算されるのは，スタック・サイズが最大となる関数だけです。
W0594161	［メッセージ］	サイズ設定情報が文字数限界 (5,119) を越えています。 呼び出し関数を減らしてください。
	［説明］	サイズ設定情報が文字数限界（5,119）を越えています。 呼び出し関数の指定を減らしてください。呼び出す関数すべてを登録しても，スタック・サイズに加算されるのは，スタック・サイズが最大となる関数だけです。 5,119 文字には，パラメータ名とセパレータ（“,”）を含みます。
W0594162	［メッセージ］	加算サイズの値（*値*）が不正です。
	［説明］	加算サイズに不正な値が指定されました。 加算サイズは，10 進数，または “0x” か “0X” で始まる 16 進数で指定可能な値を指定してください。 指定可能な値は次のとおりです。 0 ～ 2,147,483,647
W0594163	［メッセージ］	再帰回数の値（*値*）が不正です。
	［説明］	再帰回数に不正な値が指定されました。 10 進数，または “0x” か “0X” で始まる 16 進数で正の値を指定してください。
W0594164	［メッセージ］	加算サイズが大きすぎます。 *値* 以下の値を指定してください。
	［説明］	加算サイズに限界値を越える値が指定されました。 値を修正してください。 指定可能な値は次のとおりです。 0 ～ 2,147,483,647

W0594165	［メッセージ］	関数 （*関数名*）のスタックサイズが*値*を越えました。 再帰回数の値を小さくしてください。
	［説明］	関数のスタック使用量が，限界値を越えました。
	［対処方法］	再帰回数の値を小さくしてください。
W0594166	［メッセージ］	関数 （*関数名*）のスタックサイズが、*値*を越えました。 最大経路中の関数からスタックサイズを減らしてください。
	［説明］	関数のスタック使用量が，限界値を越えました。
	［対処方法］	最大経路中の関数からスタック・サイズを減らしてください

11. 注意事項

この章では，CC-RH を用いる際に注意すべき点について説明します。

11.1 volatile 修飾子

volatile 修飾子をつけて変数宣言すると，その変数は最適化の対象から外され，レジスタに割り付ける最適化などを行わなくなります。volatile 指定された変数に対する操作を行うときは，必ずメモリから値を読み込み，操作後にメモリへ値を書き込むコードになります。また，volatile 指定された変数のアクセス幅も変更されません。

volatile 指定されていない変数は，最適化によってレジスタに割り付けられ，その変数をメモリからロードするコードが削除されることがあります。また，volatile 指定されていない変数に同じ値を代入する場合，冗長な命令と解釈されて最適化により命令が削除されることもあります。特に周辺 I/O レジスタへアクセスする変数や，割り込み処理で値が変更される変数，また，外部から値が変更される変数に対しては，volatile 指定する必要があります。

volatile 指定すべきところで指定されていなかった場合，次の現象が起こることがあります。

- 正しい計算結果が得られない
- ループ内で変数を使っていた場合，ループから抜け出せない
- 命令の実行順序が変わる
- メモリのアクセス回数・アクセス幅が変わる

ただし，volatile 指定した変数を使用する際，ある区間でその変数の値が外部から変更されないことが自明な場合，volatile 指定されていない変数に，その値を代入してその変数を参照することにより，その変数が最適化され，実行速度が向上する可能性があります。

【volatile 指定しなかった場合のソースと出力コードの例】

“変数 a”，“変数 b”，および“変数 c”を volatile 指定しなかった場合，これらの変数がレジスタに割り付けられ，最適化されます。たとえば，この間に割り込みが入り，割り込み内で変数値を変更しても，値が反映されないことになります。

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        MOVHI   HIGHW1(#_a), R0, R6
        LD.W    LOWW(#_a)[R6], R6
        CMP     0x00000000, R6
        MOVHI   HIGHW1(#_b), R0, R6
        LD.W    LOWW(#_b)[R6], R6
        BGT     .BB1_2          ; bb3
.BB1_1:     ; bb1
        ADD     0x00000001, R6
        BR      .BB1_3          ; bb9
.BB1_2:     ; bb3
        ADD     0x00000002, R6
.BB1_3:     ; bb9
        ADD     0x00000001, R6
        MOVHI   HIGHW1(#_b), R0, R7
        ST.W    R6, LOWW(#_b)[R7]
        JMP     [R31]
```

【volatile 指定した場合のソースと出力コードの例】

“変数 a”，“変数 b”，および“変数 c”を volatile 指定した場合，これらの変数値を必ずメモリから読み込み，操作後にメモリへ書き込むコードが出力されます。たとえば，この間に割り込みが入り，割り込み内で変数値が変更されても，その変更が反映された結果を取得することができます（このような例の場合，割り込みのタイミングによっては，変数の操作区間内を割り込み禁止にするなどの処置が必要となります）。

volatile 指定をすると，メモリの読み込み／書き込み処理が入るため，volatile 指定しなかった場合よりもコード・サイズは大きくなります。

`volatile int a;`
`volatile int b;`
`void func(void){`
`  if(a <= 0){`
`  b++;`
`  } else {`
`  b+=2;`
`  }`
`  b++;`
`}`
	`_func:`
	`  MOVHI  HIGHW1(#_a), R0, R6`
	`  LD.W  LOWW(#_a)[R6], R6`
	`  CMP  0x00000000, R6`
	`  BGT  .BB1_2  ; bb3`
	`.BB1_1:  ; bb1`
	`  MOVHI  HIGHW1(#_b), R0, R6`
	`  LD.W  LOWW(#_b)[R6], R6`
	`  ADD  0x00000001, R6`
	`  BR  .BB1_3  ; bb9`
	`.BB1_2:  ; bb3`
	`  MOVHI  HIGHW1(#_b), R0, R6`
	`  LD.W  LOWW(#_b)[R6], R6`
	`  ADD  0x00000002, R6`
	`.BB1_3:  ; bb9`
	`  MOVHI  HIGHW1(#_b), R0, R7`
	`  ST.W  R6, LOWW(#_b)[R7]`
	`  LD.W  LOWW(#_b)[R7], R6`
	`  ADD  0x00000001, R6`
	`  ST.W  R6, LOWW(#_b)[R7]`
	`  JMP  [R31]`

11.2 アセンブラにおける V850E3v5 G3K コア指定（-Xcpu=g3k オプション）

下記の命令セットは記述することができません。

- ハードウエア・マルチスレッディング機能命令
- 仮想化支援機能命令
- MMU 制御命令
- SIMD 命令
- 浮動小数点演算命令

上記に示す命令セットは V850E3v5 G3K コア指定時はアセンブルエラーとし，記述した場合は次のメッセージが出力されます。

E0550269 : RH850 G3K コア指定時には記述することのできない命令が記述されています。

A. クイック・ガイド

この章では，CC-RH をより効果的に用いるためのプログラミング技法，および拡張機能の利用方法について説明します。

A.1 変数（C 言語）

この節では，変数（C 言語）について説明します。

A.1.1 短い命令長でアクセスできる領域へ配置する

CC-RH では，変数のアクセスは通常 movhi 命令（4 バイト）と ld/st 命令（4 バイト）の 2 命令（計 8 バイト）を使用しますが，#pragma section 指令を活用することで ld/st 命令（4 バイト，または 6 バイト），または sld/sst 命令（2 バイト）の 1 命令でアクセスするコードを生成します。これにより，コード・サイズを削減することができます。以下に詳細を示します。

A.1.1.1 GP 相対アクセス

GP（グローバル・ポインタ）と，ld/st 命令によってアクセスできるセクションに変数を配置することにより変数のアクセスを 1 命令で行うコードを生成します。
変数の定義／ 参照時は，#pragma section 指令を使用し，属性指定文字に gp_disp16/ gp_disp23 のいずれかを指定します。

```
#pragma section 属性指定文字
変数宣言／定義
#pragma section default
```

例 1. GP 相対で 4 バイトのロード／ストア命令によりアクセスする場合

```
#pragma section gp_disp16
int a = 1;              /*.sdata セクションに配置する */
int b;                  /*.sbss セクションに配置する */
#pragma section default
```

例 2. GP 相対で 6 バイトのロード／ストア命令によりアクセスする場合

```
#pragma section gp_disp23
int a = 1;             /*.sdata23 セクションに配置する */
int b;                 /*.sbss23 セクションに配置する */
#pragma section default
```

A.1.1.2 EP 相対アクセス

EP（エレメント・ポインタ）と，sld/sst 命令，または ld/st 命令によってアクセスできるセクションに変数を配置することによりコード・サイズを削減することができます。以下のいずれかの方法により，EP 相対でアクセスするセクションに変数を配置させることができます。

(1) -Omap/-Osmap オプション指定
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，アクセス頻度の高い外部変数に対して EP 相対アクセスするコードを出力します。

(2) #pragma section 指令
変数の定義／ 参照時は，#pragma section 指令を使用し，属性指定文字に ep_disp4/ep_disp5/ep_disp7/ep_disp8/ep_disp16/ep_disp23/ep_auto のいずれかを指定します。

```
#pragma section 属性指定文字
変数宣言／定義
#pragma section default
```

例 1. EP 相対で 2 バイトのロード／ストア命令によりアクセスする場合

```
#pragma section ep_disp4
int a = 1;              /*.tdata4 セクションに配置する */
int b;                  /*.tbss4 セクションに配置する */
#pragma section default
```

属性指定文字として，ep_disp5/ep_disp7/ep_disp8 を指定しても同様のアクセスとなります。

例 2. EP 相対で 4 バイトのロード／ストア命令によりアクセスする場合

```
#pragma section ep_disp16
int a = 1;              /*.edata セクションに配置する */
int b;                  /*.ebss セクションに配置する */
#pragma section default
```

例 3. EP 相対で 6 バイトのロード／ストア命令によりアクセスする場合

```
#pragma section ep_disp23
int a = 1;              /*.edata23 セクションに配置する */
int b;                  /*.ebss23 セクションに配置する */
#pragma section default
```

A.1.2 配置領域を変更する

デフォルトの配置セクションは，次のとおりになります

- 初期値なし変数：.bss セクション
- 初期値あり変数：.data セクション
- const 変数：.const セクション

配置する領域（セクション）を変更するには，#pragma section 指令で属性指定文字を指定します。

属性指定文字と生成されるセクションの関係は次のとおりです。

属性指定文字	初期値あり・なし	デフォルト・セクション名	セクション名変更	ベース・レジスタ	アクセス命令
r0_disp16	あり	.zdata	可	r0	ld/st 1 命令
	なし	.zbss	可	r0	ld/st 1 命令
r0_disp23	あり	.zdata23	可	r0	ld23/st23 1 命令
	なし	.zbss23	可	r0	ld23/st23 1 命令
r0_disp32	あり	.data	可	r0	movhi+ld/st 2 命令
	なし	.bss	可	r0	movhi+ld/st 2 命令
ep_auto	あり／なし	.tdata4/.tdata5/.tdata7/ .tdata8 から自動選択	不可	ep	sld/sst 1 命令

属性指定文字	初期値あり・なし	デフォルト・セクション名	セクション名変更	ベース・レジスタ	アクセス命令
ep_disp4	あり	.tdata4	可	ep	sld/sst 1 命令
	なし	.tbss4	可	ep	sld/sst 1 命令
ep_disp5	あり	.tdata5	可	ep	sld/sst 1 命令
	なし	.tbss5	可	ep	sld/sst 1 命令
ep_disp7	あり	.tdata7	可	ep	sld/sst 1 命令
	なし	.tbss7	可	ep	sld/sst 1 命令
ep_disp8	あり	.tdata8	可	ep	sld/sst 1 命令
	なし	.tbss8	可	ep	sld/sst 1 命令
ep_disp16	あり	.edata	可	ep	ld/st 1 命令
	なし	.ebss	可	ep	ld/st 1 命令
ep_disp23	あり	.edata23	可	ep	ld23/st23 1 命令
	なし	.ebss23	可	ep	ld23/st23 1 命令
gp_disp16	あり	.sdata	可	gp	ld/st 1 命令
	なし	.sbss	可	gp	ld/st 1 命令
gp_disp23	あり	.sdata23	可	gp	ld23/st23 1 命令
	なし	.sbss23	可	gp	ld23/st23 1 命令
const	あり	.const	可	r0	movhi+ld/st 2 命令
zconst	あり	.zconst	可	r0	ld/st 1 命令
zconst23	あり	.zconst23	可	r0	ld23/st23 1 命令
default	これ以降，#pragma section 指定がなかったものとみなし，デフォルトの配置を行います。				

例　　#pragma section 指令記述

```
#pragma section gp_disp16 "mysdata"
int a = 1;              /*mysdata.sdata セクションに配置する */
int b;                  /*mysdata.sbss セクションに配置する */
#pragma section default
```

なお，#pragma section 指令を使った変数に別ファイル（参照ファイル）の関数から参照する場合には，参照ファイルにも #pragma section 指令を記述し，該当変数を extern 宣言する必要があります。
定義と宣言で指定が異なり，指定したセクション属性ではアクセスできない場合に，以下のエラーを出力します。

E0562330 : Relocation size overflow : " *ファイル*"-" *セクション*"-" *オフセット* "

例 1.　　変数を定義しているファイル

```
#pragma section zconst
const unsigned char table_data[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
                                                /*.zconst セクションに配置する */
#pragma section default
```

例 2. 変数を参照するファイル

```
#pragma section zconst
extern  const unsigned char  table_data[];      /*.zconst セクションに配置する */
#pragma section default
```

なお，SH ファミリ用コンパイラの C ソース移植性を考慮して，以下のような表記も記述可能です。

例 #pragma section 指令記述

```
#pragma section mydata
int a = 1;              /*mydata.data セクションに配置する */
int b;                  /*mydata.bss セクションに配置する */
#pragma section default
```

A.1.3 -Xpreinclude オプションを使用して配置領域を変更する

-Xpreinclude オプションによって，C ソース・ファイルを変更せずに，ファイル内で宣言・定義したすべての変数を任意のセクションに配置させることが可能です。アクセス効率の良いセクションに配置させることでコード・サイズを削減することができます。

(1) #pragma section 指令を記述したヘッダ・ファイル（.h）を準備します。

例 .sdata/.sbss セクションに配置させる場合 [section.h]

```
#pragma section gp_disp16
```

(2) -Xpreinclude オプションによって，（1）で作成したヘッダ・ファイルをコンパイル単位の先頭にインクルードします。

例 セクション指定したヘッダ・ファイル名が section.h の場合

```
>ccrh main.c -Xpreinclude=section.h
```

main.c の先頭に，section.h がインクルードされているものとしてコンパイルします。

ただし，（1）で指定したセクションに変数が収まりきらない場合は，リンク時に以下エラーが出力されます。この場合は，C ソース・ファイル上で変数のセクションを変更してください。

```
E0562330 : Relocation size overflow : " ファイル"-" セクション"-" オフセット"
```

例 変数を .sdata23/.sbss23 セクションに変更する場合

```
int a = 1;              /*(1) で指定したセクションに配置する */
int b;                  /*(1) で指定したセクションに配置する */
#pragma section gp_disp23
int c = 1;              /*.sdata23 セクションに配置する */
int d;                  /*.sbss23 セクションに配置する */
#pragma section default
int e = 1;              /* デフォルトの .data セクションに配置する */
int f;                  /* デフォルトの .bss セクションに配置する */
```

A.1.4 通常時と割り込み時に使用する変数を定義する

通常時の処理と割り込みの処理の両方で使用する変数は，volatile 指定してください。

volatile 修飾子をつけて変数宣言すると，その変数は最適化の対象になりません。volatile 指定された変数に対する操作を行うときは，必ずメモリから値を読み込み，volatile 指定された変数に値を代入するときは必ずメモリへ値を書き込みます。また，volatile 指定された変数のアクセス順序やアクセス幅も変更されません。volatile 指定されていない変数は，最適化によってレジスタに割り付けられ，その変数をメモリからロードするコードが削除されることがあります。

また，volatile 指定されていない変数に同じ値を代入する場合，冗長な処理と解釈されて最適化によりコードが削除されることもあります。

例 1. volatile 指定しなかった場合のソースと出力コードイメージの例
“ 変数 a”，“ 変数 b” を volatile 指定しなかった場合，これらの変数がレジスタに割り付けられ，最適化される場合があります。たとえば，この間に割り込みが入り，割り込み内で変数値を変更しても，値が反映されないことになります。

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movhi   highw1(#_a), r0, r6
        ld.w    loww(#_a)[r6], r6
        cmp     0x00000000, r6
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        bgt     .bb1_2          ; bb3
.bb1_1:     ; bb1
        add     0x00000001, r6
        br      .bb1_3          ; bb9
.bb1_2:     ; bb3
        add     0x00000002, r6
.bb1_3:     ; bb9
        add     0x00000001, r6
        movhi   highw1(#_b), r0, r7
        st.w    r6, loww(#_b)[r7]
        jmp     [r31]
```

例 2. volatile 指定した場合のソースと出力コードの例
“ 変数 a”，“ 変数 b”，および “ 変数 c” を volatile 指定した場合，これらの変数値を必ずメモリから読み込み，値を代入するときはメモリへ書き込むコードが出力されます。たとえば，この間に割り込みが入り，割り込み内で変数値が変更されても，その変更が反映された結果を取得することができます。
volatile 指定をすると，メモリの読み込み／書き込み処理が入るため，volatile 指定しなかった場合よりもコード・サイズは大きくなります。

```
volatile int a;
volatile int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movhi   highw1(#_a), r0, r6
        ld.w    loww(#_a)[r6], r6
        cmp     0x00000000, r6
        bgt     .bb1_2          ; bb3
.bb1_1:    ; bb1
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        add     0x00000001, r6
        br      .bb1_3          ; bb9
.bb1_2:    ; bb3
        movhi   highw1(#_b), r0, r6
        ld.w    loww(#_b)[r6], r6
        add     0x00000002, r6
.bb1_3:    ; bb9
        movhi   highw1(#_b), r0, r7
        st.w    r6, loww(#_b)[r7]
        ld.w    loww(#_b)[r7], r6
        add     0x00000001, r6
        st.w    r6, loww(#_b)[r7]
        jmp     [r31]
```

A.1.5 const 定数ポインタを定義する

ポインタについては，“const” の指定場所により，異なる解釈がされます。
なお，.const セクションを .zconst セクションに割り当てるときは #pragma section zconst 指定をしてください。
.const セクションを .zconst23 セクションに割り当てるときは #pragma section zconst23 指定をしてください。

- const char *p ;
  ポインタが示すオブジェクト（*p）を書き換えできないことを示します。
  ポインタ自体（p）は書き換え可能です。
  したがって，以下のようになり，ポインタ自体は RAM（.data など）に配置されます。

```
*p = 0;      /* エラー */
p = 0;       /* 正しい */
```

- char *const p ;
  ポインタ自体（p）を書き換えできないことを示します。
  ポインタが示すオブジェクト（*p）は書き換え可能です。
  したがって，以下のようになり，ポインタ自体は ROM（.const/.zconst/.zconst23）に配置されます。

```
*p = 0;      /* 正しい */
p = 0;       /* エラー */
```

- const char *const p ;
  ポインタ自体（p），ポインタが示すオブジェクト（*p）を書き換えできないことを示します。
  したがって，以下のようになり，ポインタ自体は ROM（.const/.zconst/.zconst23）に配置されます。

```
*p = 0;      /* エラー */
p = 0;       /* エラー */
```

A.2 関　　数

この節では，関数について説明します。

A.2.1 配置領域を変更する

プログラム領域のセクション名を変更する場合は，以下のように #pragma section 指令を使用して関数を指定します。

```
#pragma section text  ["セクション名"]
```

#pragma section 指令で，任意の text 属性のセクションを作成する場合，実際に生成されるセクション名は “ 指定した文字列＋ .text” となります。
セクションの開始アドレスは，以下のように -start オプションで指定します。

```
-start=sec.text/1000
```

アドレスは 16 進数で指定してください。アドレスの指定がない場合は，0 番地から割り付けます。

A.2.2 離れた関数をコールする

C コンパイラは，関数呼び出しに jarl 命令を使用します。
しかし，jarl 命令は 22 ビット・ディスプレースメントであるため，プログラム配置によってはアドレス解決ができず，リンク時にエラーとなります。
上記エラーを解決する方法として，まずは -Xcall_jump=32 を指定していただくことで，jarl32，および jr32 命令を生成することができます。
-Xcall_jump=22 オプションを指定されている場合は，C コンパイラの -Xfar_jump オプションで，関数呼び出しをディスプレースメント幅に依存しない関数呼び出しにすることができます。
far jump を指定した関数の呼び出しに対しては，jarl 命令ではなく，jarl32，および jr32 命令が出力されます。
-Xfar_jump オプションで指定するファイルには，1 行に 1 関数を記述していきます。記述する名前は，C 言語関数名の先頭に “_（アンダースコア）” を付けた名前になります。

例　　-Xfar_jump オプションで指定するファイル

```
_func_led
_func_beep
_func_motor
    :
_func_switch
```

“_ 関数名 ” のかわりに次のように記述すると，すべての関数を far jump 呼び出しの対象にします。

```
{all_function}
```

A.2.3 アセンブラ命令の埋め込み

CC-RH では，次に示す形式において，C ソース・プログラム中にアセンブラ命令を記述できます。これは，関数そのものをアセンブラ命令とみなして，呼び出し箇所にインライン展開します。

- #pragma 指令

```
#pragma inline_asm
static int func(int a, int b) {
        /* アセンブラ命令 */
}
```

備考 1. アセンブラ埋め込み使用時には，下記の点に注意してください。

- #pragma inline_asm は，関数本体の定義の前に指定してください。
- #pragma inline_asm で指定した関数に対しても外部定義を生成します。
- アセンブラ埋め込みインライン関数内で関数の出入口で保証するレジスタを使用する場合は，アセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタの退避／復帰が必要です。
- RH850 のアセンブリ言語は，# で始まるコメントを使用できます。# コメントを使うと，プリプロセッサは前処理指令と解釈するため，アセンブリ記述関数内では # コメントを使わないでください。
- アセンブリ記述関数内にラベルを書くと，インライン展開の数だけ同一名のラベルが作られてしまいます。この場合は，アセンブリ記述のローカル・ラベルを使ってください。ローカル・ラベルはアセンブリ・ソースでは同一名ですが，アセンブラが自動的に別名に変換します。

備考 2. 「アセンブラ命令の記述」を参照してください。

A.2.4 RAM で実行する

リンク時とコピー後で，各セクションと各シンボル（r0，TP，EP，GP）の相対値を壊さなければ，ROM 内に配置されているプログラムを RAM にコピーして，RAM にてプログラムを実行することができます。コピーされるものと，されないものが存在するので，注意してください。

リセット後に，内蔵 RAM にコピーして，そのプログラムが変更されないのであれば，ROM 化の機能を使用して text セクションをパッキング対象にすることで簡単に実現できます。

A.3 変数（アセンブラ）

この節では，変数（アセンブラ）について説明します。

A.3.1 初期値なし変数を定義する

初期値なし変数領域を確保するには，初期値なしセクション中で，.ds 疑似命令を使用します。

```
[ ラベル :]   .ds      ( サイズ )
```

他ファイルからも参照可能にするには，そのラベルを .public 疑似命令で宣言する必要があります。

```
.public ラベル名 [, 絶対式 ]
```

例　　初期値なし変数定義

```
        .dseg   sdata
        .public _val0, 0x4      --_val0 を他ファイルから参照可能にします
        .public _val1, 0x2      --_val1 を他ファイルから参照可能にします
        .public _val2, 0x1      --_val2 を他ファイルから参照可能にします
        .align  4               --_val0 を 4 バイト整列します
_val0:
        .ds     (4)             --val0 は 4 バイトの領域を確保します
_val1:
        .ds     (2)             --val1 は 2 バイトの領域を確保します
_val2:
        .ds     (1)             --val2 は 1 バイトの領域を確保します
```

A.3.2 初期値あり変数を定義する

初期値あり変数領域を確保するには，初期値ありセクションの中で，.db 疑似命令 /.db2/.dhw 疑似命令 /.db4/.dw 疑似命令を使用します。

- 1 バイトの値の場合

```
[ ラベル :]  .db     値
```

- 2 バイトの場合

```
[ ラベル :]  .db2    値
```

- 4 バイトの場合

```
[ ラベル :]  .db4    値
```

他ファイルからも参照可能にするには，そのラベルを .public 疑似命令で宣言する必要があります。

```
.public ラベル名 [, 絶対式 ]
```

例　　初期値あり変数定義

```
        .dseg   sbss
        .public _val0, 0x4      --_val0 を他ファイルから参照可能にします
        .public _val1, 0x2      --_val1 を他ファイルから参照可能にします
        .public _val2, 0x1      --_val2 を他ファイルから参照可能にします
        .align  4               --_val0 を 4 バイト整列します
_val0:
        .db4    100             --_val0 は 4 バイト分領域を確保し，100 を格納します
_val1:
        .db2    10              --_val0 は 2 バイト分領域を確保し，10 を格納します
_val2:
        .db     1               --_val0 は 1 バイト分領域を確保し，1 を格納します
```

A.3.3 const 定数を定義する

const 定数を定義するには，.const/.zconst/.zconst23 セクション中で，.db 疑似命令 /.db2/.dhw 疑似命令 /.db4/.dw 疑似命令を使用します。

- 1 バイトの値の場合

```
[ ラベル:]   .db      値
```

- 2 バイトの場合

```
[ ラベル:]   .db2     値
```

- 4 バイトの場合

```
[ ラベル:]   .db4     値
```

例　　const 定数定義

```
        .cseg   const
        .public _p, 0x4         --_p を他ファイルから参照可能にします
        .align  4               --_p を 4 バイト整列します
_p:
        .db2    10              --_p は 2 バイト分領域を確保し，10 を格納します
```

改訂記録

Rev.	発行日	改定内容	
		ページ	ポイント
1.00	2015.02.01	-	初版発行

CC-RH V1.02.00 ユーザーズマニュアル
コンパイラ編

発行年月日　2015年 2月 1日　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 エレクトロニクス株式会社 〒100-0004 千代田区大手町2-6-2（日本ビル）

■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/

© 2015 Renesas Electronics Corporation. All rights reserved.
Colophon 2.0

CC-RH V1.02.00

RENESAS

ルネサス エレクトロニクス株式会社

R20UT3280JJ0100