User's Manual

CC-RL

コンパイラ

ユーザーズマニュアル

対象リビジョン
V1.00.00 - V1.13.00

対象デバイス
RL78 ファミリ

対象CPUコア
RL78-S1, RL78-S2, RL78-S3

ルネサス エレクトロニクス
www.renesas.com

Rev.1.13　2023.12

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。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、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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。
6. 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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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。
9. 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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。
10. 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。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。
11. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 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。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.5.0-1 2020.10)

本社所在地

〒 135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

このマニュアルの使い方

このマニュアルは，RL78 ファミリ用アプリケーション・システムを開発する際のコンパイラ（CC-RL）について説明します。

対象者

このマニュアルは，CC-RL を使用してアプリケーション・システムを開発するユーザを対象としています。

目的

このマニュアルは，CC-RL の持つソフトウエア機能をユーザに理解していただき，これらのデバイスを使用するシステムのハードウエア，ソフトウエア開発の参照用資料として役立つことを目的としています。

構成

このマニュアルは，大きく分けて次の内容で構成しています。

読み方

このマニュアルを読むにあたっては，電気，論理回路，マイクロコンピュータに関する一般知識が必要となります。

凡例

データ表記の重み	：	左が上位桁，右が下位桁
アクティブ・ロウの表記	：	XXX （端子，信号名称に上線）
注	：	本文中についた注の説明
注意	：	気をつけて読んでいただきたい内容
備考	：	本文中の補足説明
数の表記	：	10 進数 ... XXXX 16 進数 ... 0xXXXX

目次

1. 概　　説

本書は「RL78 ファミリ用 C コンパイラ」CC-RL V1.00 ～ V1.13 向けのユーザーズマニュアルです。
この章では，CC-RL の全体概要について説明します。

1.1 概　　要

CC-RL 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-RL は，次の特長を備えています。

(1) C90 および C99 に準拠した言語仕様
C 言語仕様は，C90 および C99 規格に準拠しています。

(2) 高度な最適化
大域最適化を含む高度な最適化を適用します。
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ます。

(3) 高い移植性
オブジェクトコード・フォーマットには業界標準である ELF を採用しています。また，デバッグ情報には業界標準フォーマットである DWARF2/3 を採用しています。

(4) 多機能性
ルネサス エレクトロニクス製ツール，およびパートナー製ツールとの連携による静的解析機能などを提供します。

1.3 著作権について

本ソフトウェアは次のソフトウェアを利用しています。
・LLVM および CLANG は University of Illinois at Urbana-Champaign が著作権を有します。
・Protocol Buffers は Google Inc. が著作権を有します。
C++ 用のライブラリは次のソフトウェアを利用しています。詳細はコンパイラに同梱されているライセンスファイルを参照してください。
・compiler_rt
・libc++
・libc++abi
・newlib
その他のソフトウェア構成物はルネサスエレクトロニクス株式会社が著作権を有します。

1.4 ライセンスについて

コンパイラのライセンスは，ライセンス・マネージャにより管理します。
ご使用のライセンスに応じて，standard 版，または professional 版として動作します。
standard 版と professional 版については「1.5 standard 版と professional 版について」を参照してください。
ライセンスが確認できない場合には，無償評価版として動作します。
無償評価版については「1.6 無償評価版について」を参照してください。
ライセンスおよびライセンス・マネージャの詳細は，ライセンス・マネージャのユーザーズマニュアルを参照してください。
CC-RL V1.04 以降では，ライセンス・マネージャは V2.00 以降のバージョンをご使用ください。

1.5 standard 版と professional 版について

コンパイラのエディションとして，standard 版と professional 版の 2 種類があります。
standard 版では，C90 および C99 規格に準拠した C 言語仕様をサポートし，組み込みプログラム記述に必要な基本機能を使用することができます。
professional 版では，standard 版に加えて，プログラムの品質向上と開発期間の短縮に貢献する付加機能を使用することができます。
professional 版の付加機能は，オプションまたは #pragma 指令により有効になります。

professional 版のみで使用可能なオプションは「表 2.2 コンパイル・オプション」，または各オプションの説明の項を参照してください
professional 版のみでサポートしている #pragma 指令は「表 4.15 サポートしている #pragma 指令」を参照してください。
professional 版のみでサポートしているライブラリは，「7.1 提供ライブラリ」を参照してください。

1.6 無償評価版について

無償評価版には試用期間（コンパイラの初回起動から 60 日）があり，試用期間内は，professional 版と同等の機能を使用することができます。
試用期間後は，professional 版の付加機能を使用できないほか，次の制限があります。

- 【V1.11 以前】ROM 領域に配置されるセクション・サイズの合計が 64K バイトまでになります。64K バイトを超えた場合はリンカエラーになります。
- 【V1.12 以降】使用可能な最適化レベルが -Onothing，-Olite のみになります。セクション・サイズの制限はありません。

無償評価版として動作している場合は最適化リンカのバージョン表記が W，有償版の場合はバージョン表記が V となります。以下に出力例を示します。

- 無償評価版のバージョン表記例
  Renesas Optimizing Linker W1.01.01 [25 Apr 2014]
- 有償版のバージョン表記例
  Renesas Optimizing Linker V1.01.01 [25 Apr 2014]

無償評価版では，以下のサービスは対象外となります。
サービスが必要な場合には，有償版の購入をご検討ください。

- 技術的なお問い合わせに対するサポート
- リビジョンアップ情報などの案内メール送信

2. コマンド・リファレンス

ここでは，CC-RL 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-RL 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（ccrl）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。また，ライブラリジェネレータ（lbgrl）を使用して，標準ライブラリを作成します。

ccrl ： コンパイル・ドライバの起動コマンド
asrl ： 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
lbgrl : ライブラリジェネレータ【V1.13 以降】

各コマンドの処理について説明します。

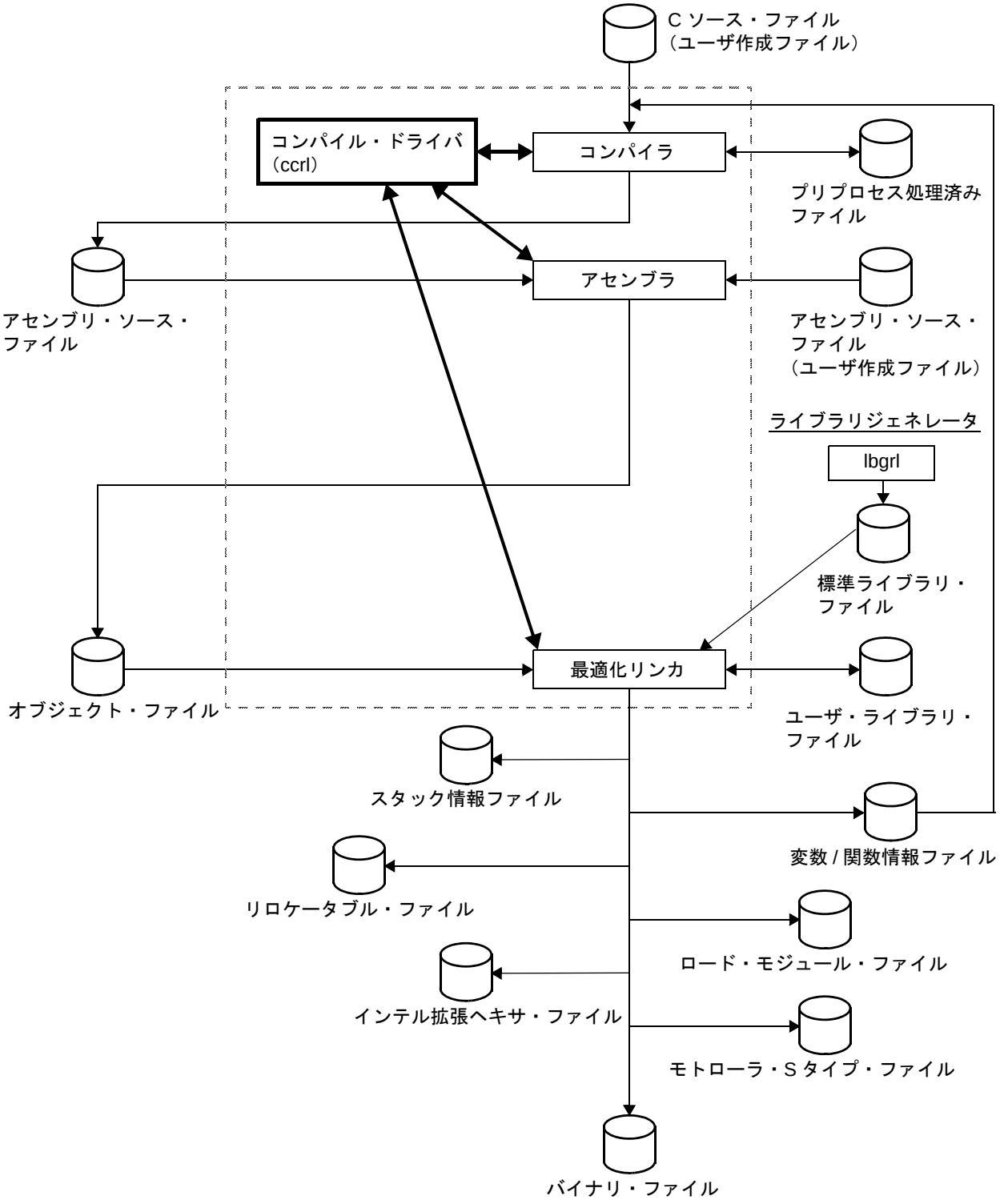
(1) コンパイラ
C ソース・プログラムに対して，プリプロセス指令の処理，コメント処理，最適化を行い，アセンブリ・ソース・プログラムを生成します。

(2) アセンブラ
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
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，組み込みアプリケーション向けの ROM イメージ作成支援，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，saddr 変数や callt 関数の宣言を記述した変数 / 関数情報ファイルの生成を行います。

(4) ライブラリジェネレータ【V1.13 以降】
標準ライブラリを生成するツールです。標準ライブラリを生成する際，任意のコンパイル・オプションを指定することができます。

図 2.1 ccrl 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 ccrl コマンドの入出力ファイルを以下に示します。

表 2.1 ccrl 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。
C++ ソース・ファイル	.cpp, .cc, .cp	入力	C++ 言語で記述したソース・ファイル ユーザ記述ファイルです。
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テキスト・ファイル -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。
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 -asmopt=-prn_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 -lnkopt=-form=library オプション指定時に出力します。
ライブラリ・バックアップ・ファイル	.lbk	出力	ライブラリジェネレータが既に存在するライブラリ・ファイルに上書きする場合に，上書き前の内容を保存しておくファイルです。
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。 -lnkopt=-form=absolute オプション指定時に出力します。 ただし，-lnkopt 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	入出力	リロケータブルなオブジェクト・ファイル -lnkopt=-form=relocate 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
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 -lnkopt=-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 -lnkopt=-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	入出力	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 -lnkopt=-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 -lnkopt=-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 -lnkopt=-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 -lnkopt=-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 -lnkopt=-stack オプション指定時に出力します。
変数 / 関数情報ファイル	.h 注 1	入出力	saddr 変数や callt 関数の宣言を記述したファイル -lnkopt=-vfinfo 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	静的解析の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。 -cref 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。 -error_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。
ツール使用情報ファイル	.ud .udm	出力	ツールの使用情報を収集するために出力するファイルです。

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（ccrl）は入力ファイルの拡張子を識別し，コンパイラ，アセンブラ，リンカを起動します。
拡張子のアルファベットは大文字 / 小文字を区別しません。

- 入力ファイルの拡張子が「.s」「.asm」のいずれかである場合，コンパイル・ドライバはそのファイルをアセンブリ言語ファイルと解釈して，アセンブラを起動します。
- 入力ファイルの拡張子が「.c」である場合，コンパイル・ドライバはそのファイルを C 言語ソース・ファイルと解釈して，コンパイラを起動します。
- 入力ファイルの拡張子が「.cpp」「.cc」「.cp」のいずれかである場合，コンパイル・ドライバはそのファイルを C++ 言語ソース・ファイルと解釈して，コンパイラを起動します。
- 入力ファイルの拡張子が「.obj」である場合，コンパイル・ドライバはそのファイルをオブジェクト・ファイルと解釈して，リンカを起動します。

これら以外の拡張子のファイルは，C ソース・ファイルと解釈してコンパイラを起動します。
C ソースファイルおよび C++ ソース・ファイルは，-lang オプションで言語規格を指定できます。

(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。

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

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

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

```
>lbgrl[ △ 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
[, ...]	:	コンマで区切って，直前のパターンを繰り返すことが可能
[: ...]	:	コロンで区切って，直前のパターンを繰り返すことが可能
string := *A*	:	*string* を *A* で置換する
string := *A* | *B* | *C*	:	*string* を *A*，*B*，*C* のいずれか 1 つで置換する

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，OS で認められるものであれば指定可能です。
  相対パス，もしくはドライブ名からはじまる絶対パスで指定します。
  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。
  この他にも，内部処理でサブコマンド・ファイルを使用するため，ファイル名やパス名の使用に注意が必要な文字があります。
  「2.4.2 サブコマンド・ファイルの使用方法」も合わせてご確認ください。
- 指定可能なファイル名の長さは，OS に依存します（Windows では、259 文字まで）。

- Windows では，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 ccrl 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 asrl 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・ファイルをカレント・フォルダに出力します。
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。

```
>ccrl file1.c file2.asm file3.obj -asmopt=-prn_path -lnkopt=-list -osample.abs -
cpu=S2 -dev=dr5f100pj.dvf
```

備考 ccrl に対して，asrl のみに指定可能なオプションを指定する場合は -asmopt 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 -lnkopt 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 ccrl 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 asrl 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・ファイルをカレント・フォルダに出力します。

```
>ccrl -c file1.c file2.asm -asmopt=-prn_path -cpu=S2 -dev=dr5f100pj.dvf
```

備考 ccrl に対して，asrl のみに指定可能なオプションを指定する場合は -asmopt 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 ccrl 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。

```
>ccrl -S file1.c -cpu=S2 -dev=dr5f100pj.dvf
```

アセンブリ・ソース・ファイル file1.asm と file2.asm を asrl 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・ファイルも出力します。

```
>asrl file1.asm -prn_path -cpu=S2 -dev=dr5f100pj.dvf
>asrl file2.asm -prn_path -cpu=S2 -dev=dr5f100pj.dvf
```

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
```

(d) 標準ライブラリを生成する場合
ライブラリジェネレータ（lbgrl）を起動して標準ライブラリを生成します。

```
>lbgrl[ △ option]...
```

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，ccrl，asrl，rlink 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，オプション指定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・ファイル内でサブコマンド・オプションを指定する場合，同名のサブコマンド・ファイルを指定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，-character_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。
  サブコマンド・ファイル中に記載された特殊文字は，ccrl コマンドに与える引数文字列から削除して，ccrl コマンドを実行します。

"（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。

```
-cpu=S2
-dev=dr5f100pj.dvf
-c
-D test
-I dir
-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 -subccomand により sub.txt を指定します。

```
>ccrl -subccomand=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。

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -c -D test -I dir -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 つのファイル名，およびパス名として，"（ダブルクォーテーション）で囲んで記述することができます。
- サブコマンド・ファイルは，システム・ロケールの文字コードで記述してください。
- 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・ファイル内にサブコマンド・オプションを指定することはできません。
  【V1.03 以前】
- サブコマンド・ファイル内でサブコマンド・オプションを指定する場合，同名のサブコマンド・ファイルを指定することはできません。
  【V1.04 以降】
- 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，ccrl 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
ライブラリ生成・フェーズ→ 「2.5.4 ライブラリジェネレータ ・オプション【V1.13.00 以降】」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。
【Professional 版のみ】の記述があるオプションは，Professional 版でのみ使用可能です。

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	ccrl のバージョン情報を表示します。
	-help	ccrl 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。
	-obj_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。
	-asm_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。
	-prep_path	プリプロセス処理後 C ソース・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。
	-g_line 【V1.02 以降】	最適化時にソース・デバッグ用の情報を強化します。
デバイス指定関係	-cpu	CPU コアの種別を指定します。
	-use_mda	乗除積和演算器を利用したコード生成を許可するか否かを指定します。
	-use_mach【V1.11.00 以降】	S3 コアが提供する積和命令 MACHU/MACH の生成をサポートします。
言語規格指定	-lang【V1.06 以降】	C ソース・ファイルおよび C++ ソース・ファイルの言語規格を指定します。
処理中断指定	-P	入力した C ソース・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。

分類	オプション	説明
プリプロセッサ制御	-D	プリプロセッサ・マクロを定義します。
	-U	-D オプションによるプリプロセッサ・マクロの定義を解除します。
	-I	ヘッダ・ファイルを検索するフォルダを指定します。
	-preinclude	コンパイル単位の先頭にインクルードするファイルを指定します。
	-preprocess	プリプロセス結果の出力を制御します。
メモリ・モデル	-memory_model	コンパイル時のメモリ・モデルの種類を指定します。
	-far_rom	ROM データのデフォルトの near/far 属性を far にします。
最適化	-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。
	-goptimize	モジュール間最適化用の情報を生成します。
エラー制御	-no_warning_num	指定した警告メッセージの出力を抑止します。
	-change_message【V1.06 以降】	メッセージレベルを変更します。
	-error_file	コンパイラのすべてのエラー・メッセージを 1 ファイルにまとめて出力します。
付加情報出力	-cref	静的解析情報ファイルを出力します。
	-pass_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。
生成コード変更	-dbl_size	double 型と long double 型の解釈を変更します。
	-signed_char	signed も unsigned も付かない char 型を符号付きとして扱います。
	-signed_bitfield	signed も unsigned も付かない型のビット・フィールドを符号付きとして扱います。
	-switch	switch 文のコード出力方式を指定します。
	-volatile	外部変数，および #pragma address 指定した変数を volatile 宣言したものとして扱います。
	-merge_string	文字列リテラルをマージします。
	-pack	構造体メンバのアライメントを 1 にします。
	-stuff 【V1.10 以降】	変数をアライメント数に応じたセクションに分けて配置します。
	-stack_protector/ -stack_protector_all 【Professional 版のみ】 【V1.02 以降】	スタック破壊検出コードを生成します。
	-control_flow_integrity 【Professional 版のみ】 【V1.06 以降】	不正な間接関数呼び出しを検出するコードを生成します。
	-insert_nop_with_label 【V1.05 以降】	ローカル・ラベルおよび nop 命令を挿入します。

分類	オプション	説明
機能拡張	-strict_std【V1.06 以降】/ -ansi【V1.05 以前】	C ソース・プログラムを -lang オプションで指定した言語規格に厳密に処理します。
	-refs_without_declaration	未宣言，または宣言がプロトタイプでない関数を参照した場合にエラーにします。
	-large_variable	変数の最大サイズを 0xffff バイトとします。
	-nest_comment	/* */ コメントのネストを可能にします。
	-character_set	日本語／中国語の文字コードを指定します。
MISRA チェック	-misra2004 【Professional 版のみ】	MISRA-C:2004 ルールによるソース・チェックを行います。
	-misra2012 【Professional 版のみ】【V1.02 以降】	MISRA-C:2012 ルールによるソース・チェックを行います。
	-ignore_files_misra 【Professional 版のみ】	MISRA-C:2004 ルールまたは MISRA-C:2012 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。
	-check_language_extension 【Professional 版のみ】	言語拡張により部分抑止される MISRA-C:2004 ルールまたは MISRA-C:2012 ルールのソース・チェックを有効にします。
	-misra_intermodule 【Professional 版のみ】【V1.08 以降】	複数ファイルにまたがる MISRA-C:2012 ルールによるソース・チェックを行います。
サブコマンド・ファイル指定	-subcommand	サブコマンド・ファイルを指定します。
アセンブラ・リンカ制御	-asmopt	アセンブル・オプションを指定します。
	-lnkopt	リンク・オプションを指定します。
	-asmcmd	アセンブラに渡すアセンブル・オプションをサブコマンド・ファイルで指定します。
	-lnkcmd	最適化リンカに渡すリンク・オプションをサブコマンド・ファイルで指定します。
	-dev	アセンブラ，および最適化リンカが使用するデバイス・ファイルを指定します。
コンパイラ移行支援機能	-convert_cc	他コンパイラ向けに書かれたプログラムの移行を支援します。
	-unaligned_pointer_for_ca78k0r【V1.06 以降】	ポインタ間接参照を 1 バイト単位でアクセスします。

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
- -help

-V

ccrl のバージョン情報を表示します。

[指定形式]

```
-V
```

- 省略時解釈
  ccrl 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]

- ccrl 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]

- ccrl のバージョン情報を標準エラー出力に出力します。

```
>ccrl -V
```

-help

ccrl のオプションの説明を表示します。

[指定形式]

```
-help
```

- 省略時解釈
  ccrl のオプションの説明を表示しません。

[詳細説明]

- ccrl 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]

- ccrl のオプションの説明を標準エラー出力に出力します。

```
>ccrl -help
```

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
- -obj_path
- -asm_path
- -prep_path

-o

出力ファイル名を指定します。

[指定形式]

```
-o △ file
```

- 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，出力先フォルダを指定しない場合，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
    最適化リンカの -output オプションに設定する出力ファイル名の指定とみなします。*file* に拡張子がない場合，出力ファイル名は最適化リンカの -form オプションによって変化します。
    詳細はリンク・オプションの説明を参照してください。
- 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。

```
>ccrl -o sample.abs -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-obj_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]

```
-obj_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。

```
>ccrl -obj_path=D:¥sample -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-asm_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]

```
-asm_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。

```
>ccrl -asm_path=D:¥sample -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-prep_path

プリプロセス処理後 C 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]

```
-prep_path[=path]
```

- 省略時解釈
  プリプロセス処理後 C ソース・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]

- -P オプション指定により生成されるプリプロセス処理後 C 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 “.i” で置き換えたファイル名でプリプロセス処理後 C 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。
  出力するプリプロセス処理後 C 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。
  出力するプリプロセス処理後 C 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 “.i” で置き換えたファイル名でプリプロセス処理後 C 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・ファイルに対するプリプロセス処理後 C ソース・ファイルのみを保存します。

[使用例]

- プリプロセス処理後 C ソース・ファイルをフォルダ D:¥sample に保存します。

```
>ccrl -prep_path=D:¥sample -P -cpu=S2 -dev=dr5f100pj.dvf 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_line 【V1.02 以降】

-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，ソース・デバッグ用の出力情報も影響を受けます。

[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。

```
>ccrl -g -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-g_line 【V1.02 以降】

最適化時にソース・デバッグ用の情報を強化します。

[指定形式]

```
-g_line
```

- 省略時解釈
  最適化時にソース・デバッグ用の情報を強化しません。

[詳細説明]

- -g オプションと同時に指定した場合にのみ有効となります。
- 最適化を行った場合に，デバッグ時により正確にソースレベルのステップ実行を行うことができるように，デバッグ情報を強化します。
- デバッグ情報量が増加し，ステップ実行が遅くなる可能性があります。

[使用例]

- 出力ファイル中の，ソース・デバッグ用の情報を強化して出力します。

```
>ccrl -g -g_line -cpu=S2 -dev=dr5f100pj.dvf main.c
```

デバイス指定関係

デバイス指定関係オプションには，次のものがあります。

- -cpu
- -use_mda
- -use_mach【V1.11.00 以降】

-cpu

CPU コアの種別を指定します。

[指定形式]

```
-cpu={S1|S2|S3}
  S1     : RL78-S1 コア
  S2     : RL78-S2 コア
  S3     : RL78-S3 コア
```

- 省略時解釈
  省略できません。指定がない場合はエラーになります。
  ただし，-V，または -help オプションを指定している場合はエラーになりません。

[詳細説明]

- CPU コアの種別を指定します。
- 指定可能な文字列以外を指定した場合は，エラーとなります。
- -cpu=S1 オプションと -dbl_size=8 オプションを同時に指定した場合は，エラーになります。
- -cpu=S2 オプションと -dbl_size=8 オプションを同時に指定した場合は，エラーになります。
- 本オプションはコア種別指定であり，演算器の有無には関係しません。
- 本オプションの指定により，-use_mda オプションの省略時解釈が異なります。
  詳細は，「-use_mda」を参照してください。
- 本オプションの指定により，-memory_model オプションの省略時解釈が異なります。
  詳細は，「-memory_model」を参照してください。

[使用例]

- CPU 種別を RL78-S2 コアに対応したコードを生成します。

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-use_mda

乗除積和演算器を利用したコード生成を許可するか否かを指定します。

[指定形式]

```
-use_mda={not_use|mda}
  not_use : コンパイラによる乗除積和演算器を利用したコード生成を抑止する
  mda     : コンパイラによる乗除積和演算器を利用したコード生成を許可する
```

- 省略時解釈
  -cpu=S1，または -cpu=S3 オプション指定時は，-use_mda=not_use オプションの指定と同じです。
  -cpu=S2 オプション指定時は，-use_mda=mda オプションの指定と同じです。

[詳細説明]

- コンパイラによる乗除積和演算器を利用したコード生成を許可するか否かを指定します。
- 下表に本オプションと -cpu オプションとを同時指定した場合の ccrl の動作を示します。

		-use_mda=	
		not_use	mda
-cpu=	S1	同時指定可能	コンパイル・エラー
	S2	同時指定可能	同時指定可能
	S3	同時指定可能	コンパイル・エラー

[注意事項]

- コンパイラ生成コード以外も制御する場合はランタイム・ライブラリの切り分けも必要です。詳細は「7.1 提供ライブラリ」を参照してください。
- 乗除算積和演算器を持たないデバイスでは -use_mda=mda を指定しないでください。

[使用例]

- 乗除積和演算器を使うコードを生成します。

```
>ccrl -use_mda=mda -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-use_mach【V1.11.00 以降】

S3 コアが提供する積和命令 MACHU/MACH の生成をサポートします。

［指定形式］

```
-use_mach={not_use|mach}
  not_use  : 命令 MACHU/MACH を使わないコードを生成する
  mach     : 命令 MACHU/MACH を使うコードを生成する
```

- 省略時解釈
  -use_mach=not_use オプションの指定と同じです。

［詳細説明］

- 積和命令 MACHU/MACH の使用の有無を指定します。
- -cpu=S1/-cpu=S2 オプションの指定時に -use_mach=mach オプションを指定するとコンパイル・エラーになります。
- -use_mach=mach オプションを指定すると，積和命令 MACHU/MACH が使用するシステム・レジスタ MACR は，関数呼出し前後，および，割込発生前後で，レジスタの値が同一であることを保証するレジスタになります。

言語規格指定

言語規格指定オプションには，次のものがあります。

- -lang【V1.06 以降】

-lang【V1.06 以降】

C ソース・ファイルおよび C++ ソース・ファイルの言語規格を指定します。

［指定形式］

```
-lang={c|c99}【V1.11 以前】
-lang={c|c99|cpp14}【V1.12 以降】
```

- 省略時解釈
  C90 規格に沿ってコンパイルします。

［詳細説明］

ソース・ファイルの言語規格を指定します。

- -lang=c オプション指定時またはオプション省略時は，C90 規格に沿ってコンパイルします。
- -lang=c99 オプション指定時は，C99 規格に沿ってコンパイルします。
- -lang=cpp14 オプション指定時は，C++14 規格に沿ってコンパイルします。【V1.12 以降】
- c,c99,cpp14 以外を指定した場合は，エラーとします。

［備考］

- -lang=cpp14 オプション指定時に，C ソース・ファイルを入力に指定した場合にはコンパイル・エラーとなります。【V1.12 以降】
- 本コンパイラは，下記の言語規格をサポートしません。
  - C90/C99 言語規格の一部のヘッダ・ファイルおよび標準ライブラリ関数
  - C99 言語規格の複素数型
  - C99 言語規格の可変長配列

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

入力した C ソース・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。
  プリプロセス処理後 C ソース・ファイルは出力しません。

[詳細説明]

- 入力した C ソース・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。
- -preprocess 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]

- 入力した C ソース・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。

```
>ccrl -P -cpu=S2 -dev=dr5f100pj.dvf 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。

```
>ccrl -S -cpu=S2 -dev=dr5f100pj.dvf 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。

```
>ccrl -c -cpu=S2 -dev=dr5f100pj.dvf 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
- -preinclude
- -preprocess

-D

プリプロセッサ・マクロを定義します。

[指定形式]

```
-D[ △ ]name[=def][,name[=def]]...
```

- 省略時解釈
  なし

[詳細説明]

- プリプロセッサ・マクロとして *name* を定義します。
- ソース・ファイルの前に，#define *name def* を記述するのと同等です。
- 本オプションで C 言語の既定義マクロ __STDC__，__LINE__，__FILE__，__DATE__，__TIME__，__RENESAS_VERSION__，__RL78__，__CCRL__，__CCRL，__RENESAS__ を再定義することはできません（ただし，-D __RL78__[=1]，-D __CCRL__[=1]，-D __CCRL[=1]，および -D __RENESAS__[=1] を除く）。
  これらの再定義は無視され警告となります。
- *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* を 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。

```
>ccrl -D sample=256 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-U

-D オプションによるプリプロセッサ・マクロの定義を解除します。

[指定形式]

```
-U[ △ ]name[,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・マクロを解除します。
- ソース・ファイルの前に，#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* 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。
- 同じプリプロセッサ・マクロに対して，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。

```
>ccrl -D TEST=XTEST -U TEST -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-I

ヘッダ・ファイルを検索するフォルダを指定します（コンパイル・フェーズ／アセンブル・フェーズ）。

[指定形式]

```
-I[ △ ]path[,path]...
```

- 省略時解釈
  ヘッダ・ファイルを ソース・ファイルのあるフォルダ，標準ヘッダ・ファイル・フォルダ（*バージョン・フォルダ*¥inc）からのみ検索します（コンパイル・フェーズ）。
  インクルード・ファイルをソース・ファイルのあるフォルダ，カレント・フォルダからのみ検索します（アセンブル・フェーズ）。

[詳細説明]

- プリプロセッサ指令 #include で読み込むヘッダ・ファイル，およびアセンブラの $INCLUDE 制御命令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。
  ヘッダ・ファイルの検索は，以下の順番で行います。

  <1> ソース・ファイルのあるフォルダ

  備考 #include でファイルをインクルードする場合は，その #include 行が書かれているファイルがあるフォルダを検索します。#include がネストしている場合はネストを逆に辿って各フォルダを検索します。

  <2> -I オプションで指定したフォルダ（指定が複数ある場合は，コマンド・ラインで指定した順（左から右の順））

  <3> 標準ヘッダ・ファイル・フォルダ

  また，アセンブラのインクルード・ファイルの検索は，以下の順番で行います。

  <1> -include オプションで指定したフォルダ（指定が複数ある場合は，コマンド・ラインで指定した順（左から右の順））

  <2> ソース・ファイルのあるフォルダ

  <3> 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]

- ヘッダ・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。

```
>ccrl -I D:¥include -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-preinclude

コンパイル単位の先頭にインクルードするファイルを指定します。

[指定形式]

```
-preinclude=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。
- *file* に指定したファイルが見つからない場合は，エラーとなります。
- *file* が相対パス指定の場合はコンパイラを起動したフォルダから検索します。

[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。

```
>ccrl -preinclude=sample.h -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-preprocess

プリプロセス結果の出力を制御します。

[指定形式]

```
-preprocess=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，警告を出力し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。

```
>ccrl -preprocess=comment,line -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

以下は上記の例と同等です。

```
>ccrl -preprocess=comment -preprocess=line -P -cpu=S2 -dev=dr5f100pj.dvf main.c
```

メモリ・モデル

メモリ・モデル・オプションには，次のものがあります。

- -memory_model
- -far_rom

-memory_model

コンパイル時のメモリ・モデルの種類を指定します。

[指定形式]

```
-memory_model={small|medium}
```

- 省略時解釈
  -cpu=S1 オプション指定時は，small を指定したものとみなします。
  -cpu=S2，または -cpu=S3 オプション指定時は，medium を指定したものとみなします。

[詳細説明]

- コンパイル時のメモリ・モデルの種類を指定します。
- 指定可能なメモリ・モデルの種類を以下に示します。

種類	メモリ・モデル	説明
small	スモール・モデル	変数，関数共にデフォルトを near とします。
medium	ミディアム・モデル	変数のデフォルトを near，関数のデフォルトを far とします。

- メモリ・モデルの種類を同時に複数指定した場合は，エラーとなります。
- 本オプションと near，far との関係については，「2.6.6 near，far との関係」を参照してください。

[使用例]

- コンパイル時のメモリ・モデルの種類としてスモール・モデルを指定します。

```
>ccrl -memory_model=small -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-far_rom

ROM データのデフォルトの near/far 属性を far にします。

[指定形式]

```
-far_rom
```

- 省略時解釈
  ROM データのデフォルトの near/far 属性は，-memory_model オプションに従います。

[詳細説明]

- メモリ・モデルで指定した ROM データに対する near/far 属性を far に設定します。
- __near/__far キーワード指定された ROM データは本オプションの影響を受けません。
- 本オプションを指定すると，ポインタの指す先が const のデータの場合，このポインタの near/far 属性は far です。しかし，const 以外のデータを指す場合は near 属性です。このため，ポインタの指す先が const データであるか否かによってポインタのサイズが異なり，C90 および C99 規格に違反しますが , そのことを承知の上でご使用ください。

```
// -far_rom を使用
      char* ptr;     // ポインタ・サイズは 2 バイト。__near の char を指すポインタ。
const char* c_ptr;   // ポインタ・サイズは 4 バイト。__far の const char を指すポインタ。
```

- const 変数へのポインタを引数に持つ標準ライブラリ関数[注]をリンクする場合，標準ヘッダ内の “#if defined(__FAR_ROM__)” により，far ポインタ用の標準ライブラリ関数に置き換えます。

  注 puts，perror，atof，atoff，strtod，strtof，atoi，atol，strtol，strtoul，bsearch，qsort，memcpy，memmove，memcmp，memchr，memset，strcpy，strncpy，strcat，strncat，strcmp，strncmp，strchr，strcspn，strpbrk，strrchr，strspn，strstr，strlen

[使用例]

- メモリ・モデルで指定した ROM データに対する near/far 属性を far に設定します。

```
>ccrl -far_rom -cpu=S2 -dev=dr5f100pj.dvf main.c
```

最適化

最適化オプションには，次のものがあります。

- -O
- -goptimize

-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]
-Oitem[=value][,item[=value]]...
```

- 省略時解釈
  -Odefault オプションを指定したものとみなします。

［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* に指定するものを以下に示します。

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 実行速度の向上を重視して，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。
default	デフォルト オブジェクト・サイズと実行速度の両方に効果のある最適化を行います。
lite 【V1.12 以降】	一部の最適化 デバッグ機能に大きく影響しない，一部の最適化を行います。
nothing	デバッグ優先の最適化 デバッグのしやすさを重視し，すべての最適化を抑止します。

なお，バージョンにより指定可能な *level* が異なります。
【V1.11 以前】 有償版・無償評価版を問わず，-Olite 以外の全ての *level* が指定可能です。
【V1.12 以降】 有償版および，試用期間内の無償評価版では全ての *level* が指定可能です。
試用期間経過後の無償評価版では -Olite と -Onothing のみが指定可能です。それ以外を指定した場合は，警告メッセージを出した上で -Olite に変更されます。

- *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 ループ文（for，while，do-while）を展開します。 *value* で，最大で何倍の展開を行うかを指定します。 *value* の値 0 は 値 1 と同じ意味となります。 *value* を省略した場合は，2 を指定したものとみなします。 本機能は -Osize，-Ospeed，または -Odefault が指定された場合に有効です。
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。

最適化項目 （*item*）	パラメータ （*value*）	説明
inline_level	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，-Osize，-Ospeed，-Odefault オプションを指定した場合は 2 を，-Olite オプションを指定した場合は 1 を指定したものとみなします。 -Osize，-Ospeed，または -Odefault オプションを指定した場合は全ての *value* が有効です。 -Olite オプションを指定した場合は 0 と 1 のみが有効です。 -Onothing オプションを指定した場合は 0 のみが有効です。
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。 100 を指定した場合はコードサイズが 100% 増加するまでインライン展開します。 *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_level=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）。
pipeline 【V1.03 以降】	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，-Osize，-Ospeed，または -Odefault を指定した場合に有効です。
tail_call	on，または off	関数末尾の関数呼び出しの br 命令置き換え 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 call 命令ではなく br 命令を生成して ret 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。
merge_files	なし	複数ファイルをマージしてコンパイル 本オプションを省略した場合は，マージせずに入力ファイル単位でコンパイルします。 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 を指定した場合は，指定した出力ファイル名となり，-o を指定しない場合は，最初に指定した C ソース・ファイル名に対して -o の省略時解釈に従ったファイル名となります。 入力ファイルが 1 ファイルの場合，および -P と同時に指定した場合は，本オプションは無効となります。 -S，または -c と同時に指定した場合，2 番目以降に指定した C ソース・ファイル名について，-o の省略時解釈に従ったファイル名の空ファイルを生成します。 -Oinline_level と同時に指定した場合，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場合，リンカ・オプション -delete，-rename，または -replace を同時に指定した場合の動作は保証しません。

最適化項目 （*item*）	パラメータ （*value*）	説明
intermodule	なし	大域最適化の実施 主な最適化内容は以下のとおりです。 - 手続き間別名解析を利用した最適化 - パラメータ，戻り値の定数伝播
whole_program	なし	入力ファイルがプログラム全体であることを仮定した最適化の実施 本オプションを省略した場合は，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。 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ている場合，その呼び出された関数からコンパイル対象ファイル内の関数が呼ばれることはない。 本オプションを指定した場合，-Ointermodule 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，-Omerge_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。
alias	ansi，または noansi	ポインタ指示先型を考慮した最適化の指定 本オプションを省略した場合は，noansi を指定したものとみなします。
same_code 【V1.02 以降】	on，または off	コンパイル単位の同一セクション内に存在する複数の同一命令列を統合し，関数化します。 *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，-Osize，-Ospeed，または -Odefault オプションを指定した場合に有効です。
branch_chaining 【V1.10 以降】	on, または off	分岐命令のコードサイズを削減する最適化の実施 コードサイズの小さな分岐命令を使用します。コードサイズの小さな分岐命令を使用するために，最終的な分岐先まで直接分岐するのではなくて，分岐先を，行き先が同じ他の分岐命令にすることがあります。 結果として，コードサイズが小さくなりますが，実行速度は低下します。 また，オプション -g_line の指定なしで本最適化を利用すると，ステップ実行時の挙動に影響がでる場合がございますのでご注意ください。 *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，-Osize または -Odefault オプションを指定した場合に有効です。
align 【V1.10 以降】	on, または off	整列条件の変更を伴う最適化の実施 たとえば構造体型の変数の中にある，連続した領域にアクセスする際に，変数の整列条件を変更した上で，複数のアクセスを 1 回に統合することで生成する命令数を少なくし，コードサイズ削減，実行速度の向上を図ります。 整列条件を変更する結果として，パディング・データの埋め込みが発生し，データを記憶する領域の使用量が増加する場合があります。 *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，-Osize，-Ospeed または -Odefault オプションを指定した場合に有効です。 -stuff オプションを同時に指定した場合，本項目は無効になります。

- 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*level* の指定により，-O*item* には以下の値を設定します。
  表中に存在しない -O*item* は，-O*level* の影響を受けません。

最適化レベル指定による最適化は下記の最適化項目と 1 対 1 に対応しているわけではありません。例えば -Odefault を指定し，各最適化項目を -Osize に合わせたとしても，-Osize と同等の最適化を実施するわけではありません。

最適化項目（*item*）	最適化レベル（*level*）				
	-Osize	-Ospeed	-Odefault	-Olite	-Onothing
unroll	1	2	1	-	-
delete_static_func	on	on	on	on	off
inline_level	3	2	3	1	0
inline_size	0	100	0	-	-
tail_call	on	on	on	off	off
pipeline	on	on	on	-	-
same_code	on	off	off	-	-
branch_chaining	on	-	on	-	-
align	on	on	off	-	-

- -O オプションの一部，または全体を省略した場合の解釈は次の通りです。
  - -O オプションを一切指定しない場合，-Odefault オプションを指定したものとみなします。
  - -O オプションの *level* パラメータを省略した場合は size を指定したものとみなします。
  - -O*level* を指定せず，-O*item* のみを指定した場合は，-O*level* には -Odefault を指定したものとみなします。

［使用例］

- オブジェクト・サイズ優先の最適化を行います。

```
>ccrl -Osize -cpu=S2 -dev=dr5f100pj.dvf main.c
```

- -O オプションの一部，または全部を省略した場合の解釈は次の通りです。

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c               # -Odefault 指定と同じ
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -O main.c            # -Osize 指定と同じ
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -Ounroll=8 main.c    # -Odefault -Ounroll=8 指定と同
                                                        じ
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -O -Ounroll=8 main.c # -Osize -Ounroll=8 指定と同じ
```

- 大域最適化を実施します。
  - 手続き間別名解析を利用した最適化

```
> ccrl -cpu=S2 im1.c -Odelete_static_func=off,intermodule
```

＜C ソース＞

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

＜出力コード＞

```
_func1@1:
    .STACK _func1@1 = 6
    movw de, ax
    push bc
    pop hl
    clrw ax
    movw [de+0x02], ax
    movw [de], ax
    onew ax
    movw [hl], ax
    clrw bc          ; a と b の指すアドレスが違うため 0 を直接代入
    clrw ax          ;
    ret
```

- パラメータ，戻り値の定数伝播

```
> ccrl -cpu=S2 im2.c -Oinline_level=1,intermodule
```

＜C ソース＞

```
static __near int func(int x, int y, int z) {
    return z-x-y;
}
int func2(void) {
    return func(3,4,8);
}
```

＜出力コード＞

```
    .SECTION .text,TEXT
_func@1:
    .STACK _func@1 = 4
    onew ax          ; 1(=8-3-4) を直接代入
    ret
    .SECTION .textf,TEXTF
_func2:
    .STACK _func2 = 4
    movw de, #0x0008
    movw bc, #0x0004
    movw ax, #0x0003
    br !_func@1
```

- ポインタ指示先の型を考慮した最適化を指定します。

```
> ccrl -cpu=S2 al1.c -Oalias=ansi
```

＜C ソース＞

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

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

```
_func:
    .STACK _func = 4
    movw de, ax
    clrw ax
    movw bc, ax
    movw !LOWW(_n+0x00002), ax
    onew ax
    movw hl, ax
    movw !LOWW(_n), ax
    onew ax
    incw ax
    movw [de], ax
    movw ax, bc     ; (A) n = 1 で代入に使用した値を再利用する
    movw !LOWW(_x+0x00002), ax
    movw ax, hl     ; (A)
    movw !LOWW(_x), ax
    ret
```

［注意］

- 最適化を適用したオブジェクト・コードを使って，ソース・レベル・デバッグを実施する場合，次のような影響があるので注意してください。
  - 最適化による式の変形（複写の伝播や共通部分式の認識など）によって，ソース・プログラム中の変数参照個所でその変数に対するリード／ライトが行われない場合があります。
  - 文に対する最適化（共通化，削除，並び替えなど）によって，ステップ実行がソース・プログラムどおりに行われない場合があります。
    また，特定の文に対してブレークポイントが設定できない場合があります。例えば，文の削除が起こった場合，その文にはブレークポイントを設定できません。
  - 文や変数に対する最適化（文の並び替え，レジスタ割付など）によって，変数の生存範囲（プログラム中でその変数を参照可能な範囲）や変数の位置（レジスタやメモリ上の位置）が変更される可能性があります。
  - インライン展開が適用された文に対するステップ実行は，インライン展開部分でなく，インライン展開元の関数内で行なわれます。
    インライン展開が適用された関数呼び出しは削除されるので，ブレークポイントは設定できません。
  - コンパイル時，デバッグ情報の処理でメモリを大量に消費するため，“out of memory” となる可能性があります。

-goptimize

モジュール間最適化用の情報を生成します。

[指定形式]

```
-goptimize
```

- 省略時解釈
  モジュール間最適化用の情報を生成しません。

[詳細説明]

- モジュール間最適化時に使用する付加情報を出力ファイル内部に生成します。
- 本オプションを指定したファイルは，リンク時にモジュール間最適化の対象になります。
  モジュール間最適化の詳細については，リンク・オプション -Optimize を参照してください。

[使用例]

- モジュール間最適化用の情報を生成します。

```
>ccrl -c -goptimize -cpu=S2 -dev=dr5f100pj.dvf main.c
```

エラー制御

エラー制御オプションには，次のものがあります。

- -no_warning_num
- -change_message【V1.06 以降】
- -error_file

-no_warning_num

指定した警告メッセージの出力を抑止します。

[指定形式]

```
-no_warning_num={num|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。
- 本オプションを複数指定した場合は，すべての指定が有効になります。
- 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。
- -change_messege オプションでエラーに変更したメッセージは，本オプションの制御の対象外になります。
- 本オプションが対象とするメッセージ番号は次の通りです。
  - W0510000 ～ W0529999，W0550000 ～ W0559999【V1.05 以前】
  - W0510000 ～ W0559999【V1.06 以降】

[使用例]

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

```
>ccrl -no_warning_num=11146,11147 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-change_message【V1.06 以降】

メッセージレベルを変更します。

[指定形式]

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

- 省略時解釈
  メッセージレベルを変更しません。

[詳細説明]

- 指定した警告メッセージをエラーメッセージに変更します。
  本オプションが対象とするメッセージ番号は，W0510000~W0549999 です。
- *num*，*num1*，*num2* には，メッセージ番号の下位 5 桁を指定します。
  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。
- メッセージ番号を省略した場合は，すべてのメッセージ番号を指定したとみなします。
- *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。
- 本オプションを複数指定した場合は，すべての指定が有効になります。

-error_file

コンパイラのすべてのエラー・メッセージを指定したファイルに出力します。

[指定形式]

```
-error_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。

```
>ccrl -error_file=err -cpu=S2 -dev=dr5f100pj.dvf main.c
```

付加情報出力

付加情報出力オプションには，次のものがあります。

- -cref
- -pass_source

-cref

静的解析情報ファイルを出力します。

[指定形式]

```
-cref=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，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。

```
>ccrl -cref=info.cref -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-pass_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]

```
-pass_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。

```
>ccrl -pass_source -S -cpu=S2 -dev=dr5f100pj.dvf main.c
```

生成コード変更

生成コード変更オプションには，次のものがあります。

- -dbl_size
- -signed_char
- -signed_bitfield
- -switch
- -volatile
- -merge_string
- -pack
- -stuff 【V1.10 以降】
- -stack_protector/-stack_protector_all 【Professional 版のみ】【V1.02 以降】
- -control_flow_integrity【Professional 版のみ】【V1.06 以降】
- -insert_nop_with_label 【V1.05 以降】

-dbl_size

double 型と long double 型の解釈を変更します。

[指定形式]

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

- 省略時解釈
  double 型，および long double 型を共に単精度浮動小数点型として扱います（-dbl_size=4 オプションの指定と同じです）。

[詳細説明]

- double 型と long double 型の解釈を変更します。
- パラメータに 4 を指定した場合，double 型，および long double 型を単精度浮動小数点型として扱います。
- パラメータに 8 を指定した場合，double 型，および long double 型を倍精度浮動小数点型として扱います。
- パラメータに 4，8 を指定しない場合は，エラーとなります。
- -cpu=S1 オプションと -dbl_size=8 オプションを同時に指定した場合は，エラーとなります。
- -cpu=S2 オプションと -dbl_size=8 オプションを同時に指定した場合は，エラーとなります。
- -dbl_size=4 オプション指定時に double 型用の標準ライブラリ関数を呼んだ場合，float 型用の標準ライブラリ関数に置き換えます。
- 本オプションは定義済みマクロに影響を与えます。

[使用例]

- double 型，および long double 型を共に float 型とみなします。

```
>ccrl -dbl_size=4 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-signed_char

signed も unsigned も付かない char 型を符号付きとして扱います。

[指定形式]

```
-signed_char
```

- 省略時解釈
  signed も unsigned も付かない char 型を符号なしとして扱います。

[詳細説明]

- signed も unsigned も付かない char 型を汎整数拡張する際に，符号付きとして扱います。
- 本オプションは定義済みマクロに影響を与えます。
- このオプションは，ビットフィールドには作用しません。
  ビットフィールドについては -signed_bitfield オプションを使用してください。

[使用例]

- signed も unsigned も付かない char 型を符号付きとして扱います。

```
>ccrl -signed_char -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-signed_bitfield

signed も unsigned も付かない型のビット・フィールドを符号付きとして扱います。

[指定形式]

```
-signed_bitfield
```

- 省略時解釈
  signed も unsigned も付かない型のビット・フィールドを符号なしとして扱います。

[詳細説明]

- signed も unsigned も付かない型のビット・フィールドを符号付きとして扱います。
- 本オプションは定義済みマクロに影響を与えます。

[使用例]

- signed も unsigned も付かない型のビット・フィールドを符号付きとして扱います。

```
>ccrl -signed_bitfield -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-switch

switch 文のコード出力形式を指定します。

[指定形式]

```
-switch={ifelse|binary|abs_table|rel_table}
```

- 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。
- 指定可能なパラメータを以下に示します。
  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 ラベルを 1 つずつ比較する形式で出力します。 case 文の数が少ないときに，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・アルゴリズムを用いて合致する 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。
abs_table rel_table	switch 文の case 分岐テーブルを用いる方式で出力します。 case 値を基にインデックス化したテーブルを参照し，case 値から各 case ラベルの位置を取得して分岐します。どの case ラベルにも同じくらい速く分岐します，case 値が連続していない場合はテーブル内に無駄な領域が生じます。 case 値の最大値と最小値の差が 8192 を超える switch 文に対しては，本オプションを無視して省略時解釈のとおりに処理します。 abs_table を指定した場合，テーブルには各 case ラベル位置の絶対アドレスを登録します。 rel_table を指定した場合，テーブルには分岐命令から各 case ラベル位置までの相対距離を登録します。ただし，相対距離が 64K バイトを超える場合にはリンク・エラーとなります。 switch 文を持つ関数が near 領域へ配置する関数である場合は，どちらのパラメータを使用していても，絶対アドレスによるテーブルを用いたコードを生成します。

- ミラー領域がないデバイスでは -far_rom オプションを指定してください。
- パラメータを省略した場合は，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。

```
>ccrl -switch=binary -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-volatile

外部変数，および #pragma address 指定した変数を volatile 宣言したものとして扱います。

［指定形式］

```
-volatile
```

- 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，および #pragma address 指定した外部変数を volatile 宣言したものとして扱います。
  外部変数，および #pragma address 指定した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，および #pragma address 指定した外部変数を volatile 宣言したものとして扱います。

```
>ccrl -volatile -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-merge_string

文字列リテラルを 1 つの領域に割り付けます。

[指定形式]

```
-merge_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 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。

[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。

```
>ccrl -merge_string -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-pack

構造体のパッキングを行います。

[指定形式]

```
-pack
```

- 省略時解釈
  構造体のパッキングを行いません。

[詳細説明]

- 構造体メンバのアライメントを 1 にします。
- 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，1 バイトでのアライメントに詰めてコード生成を行います。
- 本オプションを指定した C ソース・ファイルと，指定しなかった C ソース・ファイルが混在した場合，動作は保証しません。
- 本オプション指定によって整列条件が 2 バイトから 1 バイトに変更となった構造体，共用体，または，それらのメンバのアドレスを，標準ライブラリ関数の実引数として渡した場合，動作は保証しません。
- 本オプション指定によって整列条件が 2 バイトから 1 バイトに変更となった構造体，または共用体メンバのアドレスを，整列条件が 2 バイトである型のポインタに渡して間接参照した場合，動作は保証しません。

[使用例]

- 構造体のパッキングを行います。

```
>ccrl -pack -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-stuff 【V1.10 以降】

変数をアライメント数に応じたセクションに分けて配置します。

[指定形式]

```
-stuff[=< 変数種別 >[,...]]
< 変数種別 > : { bss | data | const }
```

- 省略時解釈
  セクションを分けずに変数を配置します。

[詳細説明]

- stuff オプションを指定した場合，指定した < *変数種別* > に属する変数をアライメント数に応じたセクションに分けて配置します。
- bss 指定は初期値なし変数を，data 指定は初期値あり変数を，const 指定は const 変数を対象とします。
- < *変数種別* > を省略した場合は，全ての種別の変数が対象となります。
- 本オプションを複数回指定した場合，指定した全ての種別の変数が対象となります。
- 同じ変数種別を複数回指定した場合，１回指定した場合と同じ意味になります。このとき，警告を出力しません。
- < *変数種別* > に bss, data, const 以外を指定した場合はエラーとします。
- 変数の出力先はセクション名に "_< *アライメント数* >" を付加したセクションとなります。
  ただし，アライメント数が 2 の場合は，"_2" は付加されません。
  例）
  変数のアライメント数が 2：.bss
  変数のアライメント数が 1：.bss_1

[使用例]

```
// near 領域
const char  __near c_n = 1;
const short __near s_n = 2;
const long  __near l_n = 3;
// far 領域
const char  __far c_f = 1;
const short __far s_f = 2;
const long  __far l_f = 3;
```

デフォルト	-stuff 指定
.SECTION .const,CONST _c_n: .DB 0x01 .ALIGN 2 _s_n: .DB2 0x0002 .ALIGN 2 _l_n: .DB4 0x00000003 .SECTION .constf,CONSTF _c_f: .DB 0x01 .ALIGN 2 _s_f: .DB2 0x0002 .ALIGN 2 _l_f: .DB4 0x00000003	.SECTION .const_1,CONST,align=1 _c_n: .DB 0x01 .SECTION .const,CONST .ALIGN 2 _s_n: .DB2 0x0002 .ALIGN 2 _l_n: .DB4 0x00000003 .SECTION .constf_1,CONSTF,align=1 _c_f: .DB 0x01 .SECTION .constf,CONSTF .ALIGN 2 _s_f: .DB2 0x0002 .ALIGN 2 _l_f: .DB4 0x00000003

［備考］

- 各セクション名は，次のオプションまたは #pragma section での指定を反映します。
  -memory_model, -far_rom

-stack_protector/-stack_protector_all 【Professional 版のみ】 【V1.02 以降】

スタック破壊検出コードを生成します。

[指定形式]

```
-stack_protector[=num]
-stack_protector_all[=num]
```

- 省略時解釈
  スタック破壊検出コードを生成しません。

[詳細説明]

- 関数の入口・出口にスタック破壊検出コードを生成します。スタック破壊検出コードとは次に示す 3 つの処理を実行するための命令列を指します。
  (1) 関数の入口で，ローカル変数領域の直前（0xFFFFF 番地に向かう方向）に 2 バイトの領域を確保し，その領域に *num* で指定した値を格納します。
  (2) 関数の出口で，*num* を格納した 2 バイトの領域が書き換わっていないことをチェックします。
  (3) (2) で書き換わっている場合には，スタックが破壊されたとして __stack_chk_fail 関数を呼び出します。
- *num* には 0 から 65535 までの 10 進数または 16 進数の整数値を指定します。*num* の指定を省略した場合には，コンパイラが自動的に数値を指定します。
- __stack_chk_fail 関数はユーザが定義する必要があり，スタックの破壊検出時に実行する処理を記述します。
- __stack_chk_fail 関数を定義する際には，次の項目に注意してください。
  - 戻り値および引数の型を void 型とし，far 領域に配置してください。
  - static 関数にしないでください。
  - 通常の関数のように呼び出すことは禁止します。
  - __stack_chk_fail 関数は，オプション -stack_protector，-stack_protector_all と #pragma stack_protector に関わらずスタック破壊検出コードの生成の対象にはなりません。
  - 関数内では abort() を呼び出してプログラムを終了させるなど，呼び出し元であるスタックの破壊を検出した関数にリターンしないようにしてください。
  - __stack_chk_fail 関数から別の関数を呼び出す場合は，呼び出した先の関数内で再帰的にスタックの破壊を検出しないように注意してください。
- -stack_protector を指定すると，構造体，共用体または配列のローカル変数を持つ関数があり，コンパイラがそれらの変数に対して 8 バイトより大きな領域をスタックに確保した場合，この関数はスタック破壊検出コードを生成する対象となります。-stack_protector_all を指定した場合には全ての関数に対してスタック破壊検出コードを生成します。
- 本オプションと #pragma stack_protector とを同時に使用した場合は，#pragma stack_protector の指定が有効になります。
- 以下が指定された関数は，スタック破壊検出コードを生成しません。
  #pragma inline，__inline キーワード，#pragma inline_asm，#pragma no_stack_protector，#pragma rtos_interrupt，#pragma rtos_task

［例］

- ＜C ソース＞

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

void f1()    // スタックが破壊されるプログラムの例
{
    volatile char str[10];
    int i;
    for (i = 0; i <= 10; i++){
      str[i] = i;    // i=10 の場合にスタックが破壊される
    }
}

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

- ＜出力コード＞
  -stack_protector=0x1234 を指定してコンパイルした場合

```
_f1:
        .STACK _f1 = 16
        movw hl, #0x1234      ; 指定した num をスタックの領域へ格納する
        push hl
        subw sp, #0x0A
        mov h, #0x00
        movw bc, #0x000B
        movw ax, sp
        movw de, ax
.BB@LABEL@1_1:  ; bb
        mov a, h
        mov [de], a
        inc a
        mov h, a
        movw ax, bc
        addw ax, #0xFFFF
        movw bc, ax
        incw de
        bnz $.BB@LABEL@1_1
.BB@LABEL@1_2:  ; return
        addw sp, #0x0A
        pop ax                ; 関数の入口で num を格納した位置から pop 命令によりロードし，
        cmpw ax, #0x1234      ; オプションで指定した 0x1234 と比較する
        sknz                  ; 異なっている場合には，.BB@LABEL@1_4 へ分岐する
.BB@LABEL@1_3:  ; return
        ret
.BB@LABEL@1_4:  ; return
        br $___stack_chk_fail  ; __stack_chk_fail を呼び出す

___stack_chk_fail:
        .STACK ___stack_chk_fail = 4
        movw de, #SMRLW(.STR@1)
        mov a, #0x0F
        call !!_printf
        br !!_abort
        .SECTION .const,CONST
.STR@1:
        .DB 0x73
        .DB 0x74
        .DB 0x61
        .DB 0x63
        .DB 0x6B
        .DB 0x20
        .DB 0x69
        .DB 0x73
        .DB 0x20
        .DB 0x62
        .DB 0x72
        .DB 0x6F
        .DB 0x6B
        .DB 0x65
        .DB 0x6E
        .DB 0x21
        .DS (1)
```

-control_flow_integrity【Professional 版のみ】【V1.06 以降】

不正な間接関数呼び出しを検出するコードを生成します。

[指定形式]

```
-control_flow_integrity
```

- 省略時解釈
  不正な間接関数呼び出しを検出するコードを生成しません。

[詳細説明]

- 不正な間接関数呼び出しを検出するコードを生成します。
  本オプションを指定すると，C ソース・ファイル内に次の処理を行うコードを生成します。
  (1) 関数の間接呼び出しが行われる直前に，間接呼び出し先のアドレスを引数に持つチェック関数 __control_flow_integrity を呼び出します。
  (2) チェック関数内で，引数のアドレスが，間接呼び出しされる可能性のある関数アドレスのリスト（以降，関数リストと呼びます）の中に含まれるかをチェックし，含まれていない場合は，不正な関数呼び出しとみなして，__control_flow_chk_fail 関数を呼び出します。
  このように，関数呼び出しをはじめとして，プログラムの流れ（制御フロー）を変える処理の整合性をチェックすることを，Control Flow Integrity（CFI）と呼びます。
- チェック関数は下記のように定義され，ライブラリ関数として提供しています。
  void __far __control_flow_integrity(void __far *addr);
  チェック関数を通常の関数のように呼び出すことは禁止します。
- コンパイラは，間接呼び出しされる可能性のある関数の情報を C ソース・ファイルから自動で抽出します。リンカがその情報を統合して関数リストを作成します。リンカで関数リストを作成するにはリンク・オプション -CFI の指定が必要です。
  詳細は「2.5.3 リンク・オプション」を参照してください。
- __control_flow_chk_fail 関数には不正な間接関数呼び出しの検出時に実行する処理を記述します。
  この関数はユーザが定義する必要があります。
  __control_flow_chk_fail 関数を定義する際には，次の項目に注意してください。
  - 戻り値および引数の型を void 型とし，far 領域に配置してください。
  - static 関数にしないでください
  - 通常の関数のように呼び出すことは禁止します。
  - __control_flow_chk_fail 関数は，不正な間接関数呼び出しを検出するコードの生成の対象になりません。
  - __control_flow_chk_fail 関数内では abort() を呼び出してプログラムを終了するなど，チェック関数に戻らないように注意してください。

［例］

- ＜C ソース＞

```
#include <stdlib.h>

int glb;

void __control_flow_chk_fail(void)
{
    abort();
}

void func1(void) // 関数リストに追加される
{
    ++glb;
}

void func2(void) // 関数リストに追加されない
{
    --glb;
}

void (*pf)(void) = func1;

void main(void)
{
    pf(); // func1 関数の間接呼び出し
    func2();
}
```

- ＜出力コード＞
  -cpu=S2 -S -control_flow_integrity を指定してコンパイルした場合

```
___control_flow_chk_fail:
        .STACK ___control_flow_chk_fail = 4
        br !!_abort
_func1:
        .STACK _func1 = 4
        incw !LOWW(_glb)
        ret
_func2:
        .STACK _func2 = 4
        decw !LOWW(_glb)
        ret
_main:
        .STACK _main = 8
        subw sp, #0x04
        movw de, !LOWW(_pf)
        movw ax, de
        movw [sp+0x02], ax
        mov a, !LOWW(_pf+0x00002)
        mov [sp+0x00], a
        call !!___control_flow_integrity ; チェック関数の呼び出し
        mov a, [sp+0x00]
        mov cs, a
        movw ax, [sp+0x02]
        movw hl, ax
        call hl ; func1 関数の間接呼び出し
        call $!_func2 ; func2 関数の直接呼び出し
        addw sp, #0x04
        ret
        .SECTION .bss,BSS
        .ALIGN 2
_glb:
        .DS (2)
        .SECTION .data,DATA
        .ALIGN 2
_pf:
        .DB2 LOWW(_func1)
        .DB LOW(HIGHW(_func1))
        .DB 0x00
```

-insert_nop_with_label 【V1.05 以降】

ローカル・ラベルおよび nop 命令を挿入します。

［指定形式］

```
-insert_nop_with_label=file,line,label
```

- 省略時解釈
  ローカル・ラベルおよび nop 命令を挿入しません。

［詳細説明］

- ソース・デバッグ情報の出力を元にして，指定した位置にローカル・ラベルと nop 命令を挿入します。
- 本オプション指定時は，-g が有効になります。
- 本機能は CS+ または e2studio 経由での使用が前提であり，ユーザは直接使用しません。

機能拡張

機能拡張オプションには，次のものがあります。

- -strict_std【V1.06 以降】/-ansi【V1.05 以前】
- -refs_without_declaration
- -large_variable
- -nest_comment
- -character_set

-strict_std【V1.06 以降】/-ansi【V1.05 以前】

C ソース・プログラムを言語規格に厳密に処理します。

［指定形式］

```
-strict_std【V1.06 以降】
-ansi       【V1.05 以前】
```

- 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。C90 では，C99 で追加された仕様の一部も受容します。

［詳細説明］

- C ソース・プログラムを -lang オプションで指定した言語規格に厳密に処理し，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。
- 言語規格に厳密なコンパイル時の処理は，以下のようになります。
  - C90 準拠時
    - _Bool 型
      エラーとなります。
    - long long 型
      エラーとなります。
    - # 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。
    - 型変換
      関数ポインタを void ポインタへ代入するなどの処理がエラーとなります。
    - 2 進定数
      エラーとなります【V1.06 以降】
  - C99 準拠時【V1.06 以降】
    - # 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。
    - 型変換
      関数ポインタを void ポインタへ代入するなどの処理がエラーとなります。
    - 2 進定数
      エラーとなります。

-refs_without_declaration

宣言がない関数を呼び出す場合，または古いスタイル（K&R）の宣言が書かれた関数呼び出しをエラーとします。

[指定形式]

```
-refs_without_declaration
```

- 省略時解釈
  宣言がない関数を呼び出す場合，または古いスタイル（K&R）の宣言が書かれた関数を呼び出す場合にメッセージを出力しません。

[詳細説明]

- 宣言がない関数を呼び出す場合，または古いスタイル（K&R）の宣言が書かれた関数を呼び出す場合にエラーとします。

[使用例]

- 宣言がない関数を呼び出す場合，または古いスタイル（K&R）の宣言が書かれた関数を呼び出す場合にエラーとします。

```
>ccrl -refs_without_declaration -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-large_variable

変数の最大サイズを 0xffff バイトとします。

[指定形式]

```
-large_variable
```

- 省略時解釈
  変数の最大サイズを 0x7fff バイトとします。
  0x7fff バイトを超えるサイズの変数宣言はエラーとなります。

[詳細説明]

- 変数の最大サイズを 0x7fff バイトから 0xffff バイトに変更します。
- 0xffff バイトを超えるサイズの変数宣言はエラーとなります。
- 本オプションを使用してポインタ減算の結果が signed int で表現可能な値の範囲を超える場合，ptrdiff_t(signed int) では値を正しく表現できません。したがって，本オプション使用時は，ポインタ演算の結果に対して注意が必要です。

[使用例]

- 変数の最大サイズを 0xffff バイトとします。

```
>ccrl -large_variable -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-nest_comment

/* */ コメントのネストを可能にします。

[指定形式]

```
-nest_comment
```

- 省略時解釈
  /* */ コメントのネストは警告となります。

[詳細説明]

- /* */ コメントのネストを可能にします。

```
/*
  /*
    ネスト
  */
*/
```

[使用例]

- /* */ コメントのネストを可能にします。

```
>ccrl -nest_comment -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-character_set

日本語／中国語の文字コードを指定します。

[指定形式]

```
-character_set={none|sjis|euc_jp|utf8|big5|gbk}
```

- 省略時解釈
  日本語 OS の場合は，本オプションのパラメータに sjis を指定したものとして扱います。それ以外の OS の場合は，none を指定したものとして扱います。

[詳細説明]

- 入力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。
- パラメータ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，入力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処理しません
euc_jp	EUC（日本語）
sjis	SJIS
utf8	UTF-8
big5	繁体字中国語
gbk	簡体字中国語

- パラメータを省略した場合は，エラーとなります。

[使用例]

- 入力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。

```
>ccrl -character_set=euc_jp -cpu=S2 -dev=dr5f100pj.dvf main.c
```

MISRA チェック

MISRA チェック・オプションには，次のものがあります。

- -misra2004 【Professional 版のみ】
- -misra2012 【Professional 版のみ】【V1.02 以降】
- -ignore_files_misra 【Professional 版のみ】
- -check_language_extension 【Professional 版のみ】
- -misra_intermodule 【Professional 版のみ】【V1.08 以降】

-misra2004 【Professional 版のみ】

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］

```
-misra2004=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 15.2 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。

```
>ccrl -misra2004=apply=5.2,5.3,5.4 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

［注意］

- 本オプションを Standard 版コンパイラで指定するとエラーとなります。
- MISRA-C:2012 ルールによるソース・チェックを同時に行うことはできません。
- 本オプションは，-lang=c99 オプションを指定した場合は無効となります。

-misra2012 【Professional 版のみ】 【V1.02 以降】

MISRA-C:2012 ルールによるソース・チェックを行います。

[指定形式]

```
-misra2012=item[=value]
```

- 省略時解釈
  MISRA-C:2012 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:2012 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。
  なお，ルールの分類が “mandatory” になっているルールは，以下の指定に関わらず必ずチェック対象となります。

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 “mandatory” および “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 “mandatory” および “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。【V1.09】
  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 2.6 2.7
  3.1 3.2
  4.1 4.2
  5.1 5.2 5.3 5.4 5.5 5.6 5.7 5.8 5.9
  6.1 6.2
  7.1 7.2 7.3 7.4
  8.1 8.2 8.3 8.4 8.5 8.6 8.8 8.9 8.11 8.12 8.13 8.14
  9.1 9.2 9.3 9.4 9.5
  10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8
  11.1 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9
  12.1 12.2 12.3 12.4 12.5
  13.1 13.2 13.3 13.4 13.5 13.6
  14.2 14.3 14.4
  15.1 15.2 15.3 15.4 15.5 15.6 15.7

16.1 16.2 16.3 16.4 16.5 16.6 16.7
17.1 17.3 17.4 17.5 17.6 17.7 17.8
18.4 18.5 18.7
19.2
20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12 20.13 20.14
21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 21.10 21.11 21.12 21.13 21.15 21.16

- *item* を省略した場合は，エラーとなります。

［使用例］

- MISRA-C:2012 ルール番号 5.2，5.3 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。
  分類が “mandatory” のルールもチェック対象になります。

```
>ccrl -misra2012=apply=5.2,5.3 -cpu=S2 -dev=dr5f100pj.dvf main.c
```

［注意］

- 本オプションを Standard 版コンパイラで指定するとエラーとなります。
- MISRA-C:2004 ルールによるソース・チェックを同時に行うことはできません。

-ignore_files_misra 【Professional 版のみ】

MISRA-C:2004 ルールまたは MISRA-C:2012 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]

```
-ignore_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 ルールまたは MISRA-C:2012 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，-misra2004 または -misra2012 オプション指定時のみ有効です。
  -misra2004 または -misra2012 オプションを指定しない場合は，警告を出力し本オプションを無視します。

[使用例]

- sample.h 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。

```
>ccrl -misra2004=all -ignore_files_misra=sample.h -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[注意]

- 本オプションを Standard 版コンパイラで指定するとエラーとなります。

-check_language_extension 【Professional 版のみ】

言語拡張により部分抑止される MISRA-C:2004 ルールまたは MISRA-C:2012 ルールのソース・チェックを有効にします。

[指定形式]

```
-check_language_extension
```

- 省略時解釈
  言語拡張により部分抑止される MISRA-C:2004 ルールまたは MISRA-C:2012 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 ルールまたは MISRA-C:2012 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，-misra2004 または -misra2012 オプション指定時のみ有効です。
  -misra2004 または -misra2012 オプションを指定しない場合は，警告を出力し本オプションを無視します。

[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。

```
>ccrl -misra2004=all -check_language_extension -cpu=S2 -dev=dr5f100pj.dvf main.c
```

[注意]

- 本オプションを Standard 版コンパイラで指定するとエラーとなります。

-misra_intermodule 【Professional 版のみ】 【V1.08 以降】

複数ファイルにまたがる MISRA-C:2012 ルールのソース・チェックを行います。

[指定形式]

```
-misra_intermodule=file
```

- 省略時解釈
  なし（複数ファイルにまたがる MISRA-C:2012 ルールのソース・チェックを行いません）

[詳細説明]

- 複数ファイルのシンボル情報を *file* に収集して，複数ファイルにまたがる MISRA-C:2012 ルールのソース・チェックを行います。*file* が存在しない場合は新規作成し，*file* が存在する場合は追加保存します 。
- 本オプションは -misra2012 オプション指定時のみ有効です。-misra2012 オプションが指定されていない場合は，警告を出力し本オプションを無視します。
- *file* を省略した場合はエラーとなります。
- 解析の有効範囲が「システム」であるルールに対して本オプションを適用します。本オプションでチェック可能な MISRA-C:2012 ルールは以下の通りです。【V1.08】
  5.1 5.6 5.7 5.8 5.9
  8.3 8.5 8.6

[使用例]

- a.c，b.c，c.c に対して複数ファイルにまたがる MISRA-C:2012 ルールのソース・チェックを行います。

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf -misra2012=all -misra_intermodule=test.mi a.c b.c
c.c
```

[注意]

- *file* の拡張子に .{c|a|f} は指定できません。指定した場合はエラーとなります。また，*file* が他の入出力ファイルと重複する場合は，動作を保証しません。
- チェック対象のファイル数が多く，*file* に入るシンボル情報が膨大になると，コンパイル速度が遅くなります。
- *file* 作成後にソース・ファイルを修正した場合は，再コンパイルすることで *file* の情報が更新されます。ソース・ファイルを削除するかソース・ファイル名を変更した場合は *file* の情報を更新できないため，*file* を削除して MISRA-C:2012 ルールのソース・チェックをやり直してください。
- 本オプションを Standard 版コンパイラで指定するとエラーとなります。

[備考]

- 本オプションは，パラレル・ビルドなど並列でコンパイルする場合には，正しいチェックができません。並列コンパイルはせずにご指定してください。

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

サブコマンド・ファイルを指定します。

[指定形式]

```
-subcommand=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。

```
>ccrl -subcommand=command.txt -cpu=S2 -dev=dr5f100pj.dvf
```

アセンブラ・リンカ制御

アセンブラ・リンカ制御オプションには，次のものがあります。

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

-asmopt

アセンブル・オプションを指定します。

[指定形式]

```
-asmopt=arg
```

- 省略時解釈
  コンパイル・ドライバが設定したアセンブラ・オプションのみをアセンブラに渡します。

[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。
- *arg* を省略した場合は，エラーとなります。

[使用例]

- -prn_path オプションをアセンブラに渡します。

```
>ccrl -c -asmopt=-prn_path -cpu=S2 -dev=dr5f100pj.dvf main.c
```

上記の例で -asmopt で指定するオプションは以下の例と同じ効果を持ちます。

```
>ccrl -S -cpu=S2 -dev=dr5f100pj.dvf main.c
>asrl -prn_path -cpu=S2 -dev=dr5f100pj.dvf main.asm
```

-lnkopt

リンク・オプションを指定します。

[指定形式]

```
-lnkopt=arg
```

- 省略時解釈
  コンパイル・ドライバが設定したリンク・オプションのみを最適化リンカに渡します。

[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。
- *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。

```
>ccrl -lnkopt=-form=relocate -cpu=S2 -dev=dr5f100pj.dvf main.c
```

上記の例で -lnkopt で指定するオプションは以下の例と同じ効果を持ちます。

```
>ccrl -c -cpu=S2 -dev=dr5f100pj.dvf main.c
>rlink -form=relocate main.obj
```

-asmcmd

アセンブラに渡すアセンブル・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]

```
-asmcmd=file
```

- 省略時解釈
  コンパイル・ドライバが設定したアセンブラ・オプションのみをアセンブラに渡します。

[詳細説明]

- アセンブラに渡すアセンブル・オプションをサブコマンド・ファイル *file* で指定します。
- 本オプションを複数指定した場合は，すべてのサブコマンド・ファイルを有効とします。
- *file* を省略した場合は，エラーとなります。

[使用例]

- アセンブラに渡すアセンブル・オプションをサブコマンド・ファイル command_asm.txt で指定します。

```
>ccrl -asmcmd=command_asm.txt -cpu=S2 -dev=dr5f100pj.dvf
```

-lnkcmd

最適化リンカに渡すリンク・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]

```
-lnkcmd=file
```

- 省略時解釈
  コンパイル・ドライバが設定したリンク・オプションのみを最適化リンカに渡します。

[詳細説明]

- 最適化リンカに渡すリンク・オプションをサブコマンド・ファイル *file* で指定します。
- 本オプションを複数指定した場合は，すべてのサブコマンド・ファイルを有効とします。
- *file* を省略した場合は，エラーとなります。

[使用例]

- 最適化リンカに渡すリンク・オプションをサブコマンド・ファイル command_lnk.txt で指定します。

```
>ccrl -lnkcmd=command_lnk.txt -cpu=S2 -dev=dr5f100pj.dvf
```

-dev

アセンブラ，および最適化リンカが使用するデバイス・ファイルを指定します。

[指定形式]

```
-dev=file
```

- 省略時解釈
  デバイス・ファイルをアセンブラ，および最適化リンカに渡しません。

[詳細説明]

- アセンブラ，および最適化リンカが使用するデバイス・ファイル *file* を指定します。
- コンパイル時に本オプションを指定しない場合は，アセンブラ，または最適化リンカでエラーとなることがあります。
- 本オプションを複数指定した場合は，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]

- アセンブラ，および最適化リンカが使用するデバイス・ファイル DR5F100PJ.DVF を指定します。

```
>ccrl -cpu=S2 -dev=dr5f100pj.dvf main.c
```

コンパイラ移行支援機能

コンパイラ移行支援機能オプションには，次のものがあります。

- -convert_cc
- -unaligned_pointer_for_ca78k0r【V1.06 以降】

-convert_cc

他コンパイラ向けに書かれたプログラムの移行を支援します。

[指定形式]

```
-convert_cc={ca78k0r|nc30|iar}
```

- 省略時解釈
  他コンパイラ向けに書かれたプログラムの移行支援機能を無効にします。

[詳細説明]

- 他コンパイラの拡張機能を CC-RL の拡張機能に変換します。動作は CC-RL の仕様に従います。
- ANSI C 言語における未規定，未定義，処理系定義の項目については，移行前のコンパイラと同じ動作を保障しません。
- 本オプションを複数回指定した場合はコンパイル・エラーとなります。
- 本オプションにおけるコンパイラ種別が異なるオブジェクト同士をリンクする場合，動作を保証しません。
- 指定可能なパラメータを以下に示します。
  これ以外のものを指定した場合はコンパイル・エラーとなります。

パラメータ	説明
ca78k0r	CA78K0R の拡張仕様に対する移行支援機能を有効にします。
nc30	NC30 の拡張仕様に対する移行支援機能を有効にします。
iar	ICCRL78（IAR コンパイラ）の拡張仕様に対する移行支援機能を有効にします。

- 本オプションは，-lang=c99 オプションを指定した場合は無効となります。

-convert_cc=ca78k0r を指定した場合の動作は以下のとおりです。

- マクロ __CNV_CA78K0R__ を有効にします。
- #pragma に続くキーワードがすべて大文字か，すべて小文字の場合に認識します。
  キーワードに大文字と小文字が混在した場合は，未知のキーワードとして扱います。
- 拡張仕様の対応は以下のとおりです。

表 2.3 移行支援機能オプション指定時の動作（-convert_cc=ca78k0r）

ca78k0r の機能	CC-RL での機能	オプション指定時の動作
__callt, callt	__callt	-strict_std オプション非指定時に，callt キーワードを __callt に置換します。
__callf, callf	なし	サポート対象外です。 構文エラーとなります。
__sreg, sreg	__saddr	__sreg キーワードを __saddr に置換します。 -strict_std オプション非指定時に，sreg キーワードを __saddr に置換します。
__leaf, norec, noauto	なし	サポート対象外です。 構文エラーとなります。

ca78k0r の機能	CC-RL での機能	オプション指定時の動作
__boolean, boolean, bit	なし	-strict_std オプション指定時に，__boolean キーワードを char に置換します。 -strict_std オプション非指定時に，__boolean，boolean，bit キーワードを _Bool に置換します。
__interrupt __interrupt_brk	#pragma interrupt #pragma interrupt_brk	キーワードで修飾された関数に対する #pragma 指令が同じファイル内にある場合は，キーワードを削除します。そうでない場合は，キーワードを #pragma 指令に置換します。
__asm #asm 〜 #endasm	#pragma inline_asm	サポート対象外です。 __asm は通常の関数呼び出しとして扱います。 #asm，および #endasm は構文エラーとなります。
__rtos_interrupt	#pragma rtos_interrupt	キーワードで修飾された関数に対する #pragma 指令が同じファイル内にある場合は，キーワードを削除します。そうでない場合は，キーワードを #pragma 指令に置換します。
__pascal	なし	サポート対象外です。 構文エラーとなります。
__flash	なし	サポート対象外です。 構文エラーとなります。
__flashf	なし	サポート対象外です。 構文エラーとなります。
__directmap	#pragma address	キーワードを削除して新たに #pragma address を作成します。__sreg，sreg または __saddr を同時に指定した場合はコンパイル・エラーとなります。複数の変数を同一アドレスに指定した場合はエラーとなります。
__temp	なし	サポート対象外です。 構文エラーとなります。
__near, __far	__near, __far	far ポインタの演算ルールは CC-RL の仕様に従います。 関数や関数ポインタの宣言における __near, __far キーワードの記述位置は，CC-RL の仕様に従います。 CA78KOR の仕様で記述した場合は，構文エラーとなります。 far ポインタの演算ルールについては「メモリ配置領域指定（__near/__far）」の「ポインタ演算」の項を参照してください。
__mxcall	なし	サポート対象外です。 構文エラーとなります。
#pragma sfr	#include "iodefine.h"	#pragma 指令を無視し，警告メッセージを出力します。 ビットアクセスを含む SFR への参照を，iodefine.h で定義した記号定数への参照に変換します。 iodefine.h のインクルードは手動で指定してください。

ca78k0r の機能	CC-RL での機能	オプション指定時の動作
#pragma vect #pragma interrupt	#pragma interrupt #pragma interrupt_brk	CC-RL の仕様に置換します。 vect キーワードを interrupt に置換します。 割り込み要求名が BRK_I の場合は，interrupt または vect キーワードを interrupt_brk に置換します。 スタック切り替え指定がある場合は削除し，警告メッセージを出力します。 割り込み要求名は，iodefine.h で定義したアドレスに変換します。iodefine.h のインクルードは手動で指定してください。 C ソース・ファイル中に #pragma 指令のみを記述し関数宣言や関数定義がない場合は，ベクタテーブルを生成せず，リンク時にエラーとなりません。 割り込みハンドラには一つの割り込み要求名しか設定できません。
#pragma rtos_interrupt	#pragma rtos_interrupt	CC-RL の仕様に置換します。 割り込み要求名は，iodefine.h で定義したアドレスに変換します。iodefine.h のインクルードは手動で指定してください。 C ソース・ファイル中に #pragma 指令のみを記述し関数宣言や関数定義がない場合は，ベクタテーブルを生成せず，リンク時にエラーとなりません。
#pragma rtos_task	#pragma rtos_task	CC-RL の仕様に置換します。
#pragma di #pragma ei	__DI __EI	関数 DI, EI の呼び出しを __DI, __EI に置換します。
#pragma halt #pragma stop #pragma brk #pragma nop	__halt __stop __brk __nop	関数 HALT, STOP, BRK, NOP の呼び出しを __halt, __stop, __brk, __nop に置換します。
#pragma section	#pragma section	コンパイラ出力セクション名を CC-RL 仕様に従ったセクション名に置換します。アドレス指定がある場合は削除し，警告メッセージを出力します。CC-RL のセクション名に置換できない場合は，#pragma 指令を削除し警告メッセージを出力します。 CC-RL で記述可能なセクション名については，「コンパイラ出力セクション名の変更（#pragma section）」を参照してください。
#pragma name	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma rot	__rolb, __rorb, __rolw, __rorw	関数 rolb, rorb, rolw, rorw の呼び出しを __rolb, __rorb, __rolw, __rorw に置換します。
#pragma mul	__mulu, __mului, __mulsi	関数 mulu, muluw, mulsw の呼び出しを __mulu, __mului, __mulsi に置換します。
#pragma div	__divui, __remui	関数 divuw, moduw の呼び出しを __divui, __remui に置換します。
#pragma mac	__macui, __macsi	関数 macuw, macsw の呼び出しを __macui, __macsi に置換します。
#pragma bcd	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma opc	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma ext_func	なし	#pragma 指令を削除し警告メッセージを出力します。

ca78k0r の機能	CC-RL での機能	オプション指定時の動作
#pragma inline	なし	#pragma 指令に続いて改行がある場合は #pragma 指令を無視して警告メッセージを出力します。 #pragma 指令に続いて同じ行に関数名がある場合は CC-RL の #pragma inline(別機能) として処理します。
2 進定数	2 進定数	そのまま 2 進定数として処理します。
__K0R__	__RL78__	マクロが有効となります (10 進定数 1)。
__K0R__SMALL__	__RL78_SMALL__	-memory_model オプションで small を指定した場合，または，-cpu オプションで S1 を指定し -memory_model オプションを指定しなかった場合にマクロが有効となります (10 進定数 1)。
__K0R__MEDIUM__	__RL78_MEDIUM__	-memory_model オプションで medium を指定した場合，または，-cpu オプションで S1 以外を指定し -memory_model オプションを指定しなかった場合にマクロが有効となります (10 進定数 1)。
__K0R__LARGE__	なし	サポート対象外です。 ユーザ定義マクロとして扱います。
__CHAR_UNSIGNED__	__UCHAR	-signed_char オプションを指定しなかった場合にマクロが有効となります (10 進定数 1)。
__RL78_1__	__RL78_S2__	-cpu オプションで S2 を指定した場合にマクロが有効となります (10 進定数 1)。
__RL78_2__	__RL78_S3__	-cpu オプションで S3 を指定した場合にマクロが有効となります (10 進定数 1)。
__RL78_3__	__RL78_S1__	-cpu オプションで S1 を指定した場合にマクロが有効となります (10 進定数 1)。
__CA78K0R__	なし	マクロが有効となります (10 進定数 1)。
CPU マクロ	なし	サポート対象外です。 ユーザ定義マクロとして扱います。
標準ライブラリ関数 va_starttop	va_start	stdarg.h にて，va_starttop を va_start に置換します。
標準ライブラリ関数 toup，_toupper，tolow，_tolower，__putc，calloc，free，malloc，realloc，atexit，brk，sbrk，itoa，ltoa，ultoa，strbrk，strsbrk，stritoa，strltoa，strultoa，strcoll，strxfrm，matherr，_assertfail	なし	サポート対象外です。 通常の関数呼び出しとして扱います。
標準ライブラリ関数 その他	標準ライブラリ関数	CC-RL の仕様に従います。 関数や関数ポインタに対する __near, __far キーワードの記述位置は，CC-RL の仕様に従います。CA78KOR の仕様で記述した場合は，構文エラーとなります。
標準ライブラリ マクロ	標準ライブラリ マクロ	CC-RL のヘッダ・ファイルで定義するマクロと同名のマクロは，CC-RL の仕様に従います。 それ以外のマクロはサポート対象外です。ユーザ定義マクロとして扱います。

-convert_cc=nc30 を指定した場合の動作は以下のとおりです。

- マクロ __CNV_NC30__ を有効にします。
- 拡張仕様の対応は以下のとおりです。

表 2.4 移行支援機能オプション指定時の動作（-convert_cc=nc30）

nc30 の機能	CC-RL での機能	オプション指定時の動作
wchar_t 型	なし	stddef.h にて，wchar_t 型を unsigned short 型として typedef 宣言します。
接尾語なし，接尾語 l または L の 10 進定数 int long int long long int	接尾語なし，接尾語 l または L の 10 進定数 int long int long long int	CC-RL の仕様に従います。
2 進定数	2 進定数	そのまま 2 進定数として処理します。 “_” は数値と数値の間に記述できます。それ以外の場所に記述した場合は構文エラーとなります。
ワイド文字列	ワイド文字列	文字列定数とワイド文字列定数を結合する場合は，CC-RL の仕様に従います。
関数のデフォルト引数宣言	なし	サポート対象外です。 構文エラーとなります。
near，far _near，_far	__near，__far	キーワードを __near，__far に置換します。 far ポインタの演算ルールは CC-RL の仕様に従います。 far ポインタの演算ルールについては「メモリ配置領域指定（__near/__far）」の「ポインタ演算」の項を参照してください。
asm，_asm	#pragma inline_asm	サポート対象外です。 通常の関数呼び出しとして扱います。
inline，_inline	__inline	キーワードを __inline に置換します。
restrict	なし	キーワードを削除し，警告メッセージを出力します。
_ext4mptr	なし	キーワードを削除し，警告メッセージを出力します。
#pragma ROM	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma SECTION	#pragma section	コンパイラ出力セクション名を CC-RL のセクション名に置換します。 セクション種別が不適切な場合は #pragma 指令を削除し警告メッセージを出力します。 CC-RL のセクション名に置換できない場合はコンパイル・エラーとなります。
#pragma STRUCT	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma EXT4MPTR	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma ADDRESS	#pragma address	CC-RL の #pragma address として処理します。 アドレスの数値表記が CC-RL の仕様と異なる場合は，#pragma 指令を削除し警告メッセージを出力します。
#pragma BITADDRESS	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma INTCALL	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma INTERRUPT	#pragma interrupt	CC-RL の #pragma interrupt として処理します。 CC-RL と異なるフォーマットで記述した場合は，#pragma 指令を削除し警告メッセージを出力します。

nc30 の機能	CC-RL での機能	オプション指定時の動作
#pragma PARAMETER	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma SPECIAL	#pragma callt	#pragma 指令で指定した関数を callt 関数とし，警告メッセージを出力します。 呼び出し番号は無視します。
#pragma ALMHANDLER	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma CYCHANDLER	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma INTHANDLER #pragma HANDLER	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma TASK	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma __ASMMACRO	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma ASM ～ ENDASM	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma JSRA	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma JSRW	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma PAGE	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma SBDATA	なし	#pragma 指令を削除し警告メッセージを出力します。
NC30	なし	マクロが有効となります。（空白を定義）
M16C	なし	マクロが有効となります。（空白を定義）
__R8C__	なし	マクロが有効となります。（空白を定義）
__cplusplus	なし	ユーザ定義マクロとして扱います。
標準ライブラリ関数 clearerr，fgetc，getc，fgets，fread，fscanf，fputc，putc，fputs，fwrite，fflush，fprintf，vfprintf，ungetc，ferror，feof，calloc，free，malloc，realloc，mblen，mbstowcs，mbtowc，wcstombs，wctomb，strcoll，stricmp，strnicmp，strxfrm，bzero，bcopy，memicmp，localeconv，setlocale	なし	サポート対象外です。 通常の関数呼び出しとして扱います。 サポートしていないヘッダ・ファイルをインクルードしている場合はコンパイル・エラーとなります。
標準ライブラリ関数 その他	標準ライブラリ関数	CC-RL の仕様に従います。 関数や関数ポインタに対する __near, __far キーワードの記述位置は，CC-RL の仕様に従います。
標準ライブラリ マクロ	標準ライブラリ マクロ	CC-RL のヘッダ・ファイルで定義するマクロと同名のマクロは，CC-RL の仕様に従います。 それ以外のマクロはサポート対象外です。ユーザ定義マクロとして扱います。

-convert_cc=iar を指定した場合の動作は以下のとおりです。

- マクロ __CNV_IAR__ を有効にします。
- 拡張仕様の対応は以下のとおりです。

表 2.5 移行支援機能オプション指定時の動作（-convert_cc=iar）

iar の機能	CC-RL での機能	オプション指定時の動作
wchar_t 型	なし	stddef.h にて，wchar_t 型を unsigned short 型として typedef 宣言します。
ファイルスコープの匿名共用体	なし	サポート対象外です。 構文エラーとなります。
__near，__far	__near，__far	far ポインタの演算ルールは CC-RL の仕様に従います。 far ポインタの演算ルールについては「メモリ配置領域指定（__near/__far）」の「ポインタ演算」の項を参照してください。
__near_func，__far_func	__near，__far	キーワードを __near，__far に置換します。 far ポインタの演算ルールは CC-RL の仕様に従います。 far ポインタの演算ルールについては「メモリ配置領域指定（__near/__far）」の「ポインタ演算」の項を参照してください。
__interrupt	#pragma interrupt	"#pragma interrupt 関数名 " に置換します。
__monitor	なし	キーワードを削除し，警告メッセージを出力します。
__no_bit_access	なし	キーワードを削除し，警告メッセージを出力します。
__no_init	なし	キーワードを削除し，警告メッセージを出力します。
__intrinsic	なし	サポート対象外です。 構文エラーとなります。
__noreturn	なし	キーワードを削除し，警告メッセージを出力します。
__no_save	なし	キーワードを削除し，警告メッセージを出力します。
__root	なし	キーワードを削除し，警告メッセージを出力します。
__ro_placement	なし	キーワードを削除し，警告メッセージを出力します。
__sfr	なし	サポート対象外です。 構文エラーとなります。
__saddr	__saddr	そのまま __saddr キーワードとして扱います。
@ 演算子	#pragma address	サポート対象外です。 構文エラーとなります。
__segment_begin	__sectop	変換せず，エラー・メッセージを出力します。
__segment_end	__secend	変換せず，エラー・メッセージを出力します。
__segment_size	なし	エラー・メッセージを出力します。
__ALIGNOF__	なし	エラー・メッセージを出力します。
static_assert	なし	エラー・メッセージを出力します。
__break	__brk	__brk に置換します。
__disable_interrupt	__DI	__DI に置換します。
__enable_interrupt	__EI	__EI に置換します。
__get_interrupt_level	なし	通常の関数呼び出しとして扱います。

iar の機能	CC-RL での機能	オプション指定時の動作
__get_interrupt_state	なし	通常の関数呼び出しとして扱います。
__mach	なし	通常の関数呼び出しとして扱います。
__machu	なし	通常の関数呼び出しとして扱います。
__no_operation	__nop	__nop に置換します。
__set_interrupt_level	なし	通常の関数呼び出しとして扱います。
__set_interrupt_state	なし	通常の関数呼び出しとして扱います。
__stop	__stop	そのまま __stop として扱います。
#pragma vector	#pragma interrupt	"#pragma interrupt 関数名 (vect= アドレス)" に置換します。 関数名は #pragma vector に後続する関数宣言中の関数名とし，__interrupt キーワードは削除します。 __interrupt 関数が後続しない場合は，#pragma 宣言を削除します。 割り込みハンドラに複数の割り込み要求名を指定した場合，最初の割り込み要求名を設定し，2 つ目以降の割り込み要求名は警告を出して無視します。
#pragma bank	#pragma interrupt	"#pragma interrupt 関数名 (bank={RB0 | RB1 | RB2 | RB3})" に置換します。 関数名は #pragma bank に後続する関数宣言中の関数名とし，__interrupt キーワードは削除します。 移行後のレジスタ・バンクは，#pragma bank で指定した番号の先頭に "RB" を付けたものとなります。 __interrupt 関数が後続しない場合は，#pragma 宣言を削除します。
#pragma basic_template_matching	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma bitfields	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma constseg	#pragma section	#pragma 指令を削除し警告メッセージを出力します。
#pragma data_alignment	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma dataseg	#pragma section	#pragma 指令を削除し警告メッセージを出力します。
#pragma diag_default	なし	#pragma 指令が有効となります。
#pragma diag_error	なし	#pragma 指令が有効となります。
#pragma diag_remark	なし	#pragma 指令が有効となります。
#pragma diag_suppress	なし	#pragma 指令が有効となります。
#pragma diag_warning	なし	#pragma 指令が有効となります。
#pragma error	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma include_alias	なし	#pragma 指令を削除し警告メッセージを出力します。

iar の機能	CC-RL での機能	オプション指定時の動作
#pragma inline	#pragma inline / #pragma noinline	forced 指定時は #pragma inline，never 指定時は #pragma noinline に置換します。 なお，forced 指定時に必ずインライン展開されるとは限りません。 適用する関数は #pragma inline に後続する関数宣言中の関数とします。 関数宣言以外が後続する場合はエラーとなります。 何も後続しない場合は，#pragma 指令を削除し警告メッセージを出力します。 IAR 形式の #pragma inline のみ使用可能です。CC-RL 形式の #pragma inline はコンパイル・エラーとなります。
#pragma language	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma location	#pragma address	絶対アドレスを指定した場合，#pragma address に置換します。 #pragma address で用いる変数名は #pragma location に後続する変数宣言中の変数名とします。 変数宣言が後続しない場合は，#pragma 指令を削除し警告メッセージを出力します。 セグメント名はサポート対象外です。構文エラーとなります。
#pragma message	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma object_attribute	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma optimize	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma pack	なし	アライメントが 1 と判断できた場合は CC-RL の #pragma pack に，アライメントが 2 と判断できた場合は CC-RL の #pragma unpack に変換します。 それ以外のアライメント指定，アライメント以外のパラメータは無視します。
#pragma __printf_args	なし	#pragma 指令が有効となります。
#pragma required	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma rtmodel	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma __scanf_args	なし	#pragma 指令が有効となります。
#pragma segment	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma section	なし	CC-RL の #pragma section として処理します。 CC-RL と異なるフォーマットで記述された場合は #pragma 指令を無視し，警告メッセージを出力します。
#pragma STDC CX_LIMITED_RANGE	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma STDC FENV_ACCESS	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma STDC FP_CONTRACT	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma type_attribute	なし	#pragma 指令を削除し警告メッセージを出力します。
#pragma unroll	なし	#pragma 指令を削除し警告メッセージを出力します。
#warning	なし	#pragma 指令が有効となります。
_Pragma()	なし	通常の関数呼び出しとして扱います。

iar の機能	CC-RL での機能	オプション指定時の動作
__CORE__	なし	マクロが有効となります。 -cpu オプションの指定に応じて以下のいずれかの値になります。 - __RL78_0__（-cpu オプションで S1 を指定した場合） - __RL78_1__（-cpu オプションで S2 を指定した場合） - __RL78_2__（-cpu オプションで S3 を指定した場合）
__RL78_0__	__RL78_S1__	マクロが有効となります。（値は 1）
__RL78_1__	__RL78_S2__	マクロが有効となります。（値は 2）
__RL78_2__	__RL78_S3__	マクロが有効となります。（値は 3）
__CODE_MODEL__	なし	マクロが有効となります。 -memory_model オプション，または，-cpu オプションの指定に応じて以下のいずれかの値になります。 - __CODE_MODEL_NEAR__（-memory_model オプションで small を指定した場合，または，-cpu オプションで S1 を指定し -memory_model オプションを指定しなかった場合） - __CODE_MODEL_FAR__（-memory_model オプションで medium を指定した場合，または，-cpu オプションで S1 以外を指定し -memory_model オプションを指定しなかった場合）
__CODE_MODEL_NEAR__	__RL78_SMALL__	マクロが有効となります。（値は 1）
__CODE_MODEL_FAR__	__RL78_MEDIUM__	マクロが有効となります。（値は 2）
__DATA_MODEL__	なし	マクロが有効となります。 -memory_model オプション，-cpu オプションの指定にかかわらず，値は __DATA_MODEL_NEAR__ となります。
__DATA_MODEL_NEAR__	__RL78_SMALL__	マクロが有効となります。（値は 1）
__DATA_MODEL_FAR__	なし	マクロが有効となります。（値は 2）
__func__	なし	マクロが有効となります。
__FUNCTION__	なし	マクロが有効となります。
__PRETTY_FUNCTION__	なし	マクロが有効となります。
__IAR_SYSTEMS_ICC__	なし	マクロが有効となります。（値は 8）
__ICCRL78__	なし	マクロが有効となります。（値は 1）
__BUILD_NUMBER__ __cplusplus __DOUBLE__ __embedded_cplusplus __LITTLE_ENDIAN__ __SUBVERSION__ __VER__	なし	ユーザ定義マクロとして扱います。

iar の機能	CC-RL での機能	オプション指定時の動作
標準ライブラリ関数 fabsl acosl asinl atanl atan2l ceill cosl coshl expl floorl fmodl frexpl ldexpl logl log10l modfl powl sinl sinhl sqrtl tanl tanhl strtold	なし	以下の関数名に置換します。 fabs acos asin atan atan2 ceil cos cosh exp floor fmod frexp ldexp log log10 modf pow sin sinh sqrt tan tanh strtod 関数や関数ポインタに対する __near, __far キーワードの記述位置は，CC-RL の仕様に従います。
CC-RL でサポートしている標準ライブラリ関数	標準ライブラリ関数	CC-RL の仕様に従います。 関数や関数ポインタに対する __near, __far キーワードの記述位置は，CC-RL の仕様に従います。
標準ライブラリ関数 その他	なし	サポート対象外です。 通常の関数呼び出しとして扱います。 サポートしていないヘッダ・ファイルをインクルードしている場合はコンパイル・エラーとなります。
標準ライブラリ マクロ	標準ライブラリ マクロ	CC-RL のヘッダ・ファイルで定義するマクロと同名のマクロは，CC-RL の仕様に従います。 それ以外のマクロはサポート対象外です。ユーザ定義マクロとして扱います。

［使用例］

- ca78k0r 向けに書かれたプログラムの移行支援機能を有効にします。

```
>ccrl -convert_cc=ca78k0r -cpu=S2 -dev=dr5f100pj.dvf main.c
```

-unaligned_pointer_for_ca78k0r【V1.06 以降】

ポインタ間接参照を 1 バイト単位でアクセスします。

［指定形式］

```
-unaligned_pointer_for_ca78k0r
```

- 省略時解釈
  2 バイトの整列条件を持つ型へは，2 バイトアクセスで間接参照するコードを生成します。

［詳細説明］

- 本オプションは，CA78K0R からのコンパイラ移行支援が目的であり，CA78K0R で同機能を使用している場合に指定します。本オプションを指定した場合，オブジェクト・サイズが大きくなり，実行速度が低下します。
- volatile 指定がない 2 バイトの整列条件を持つ型へのポインタが奇数番地を指す可能性がある場合に，1 バイトアクセスで間接参照するコードを生成します。volatile 指定がある場合は，2 バイトの整列条件を持つ型が奇数番地を指していたとしても，2 バイトアクセスで間接参照します。
- CC-RL において構造体パッキング時に 2 バイトの整列条件がなくなるのは構造体および構造体のメンバだけで，int 型など他の型は 2 バイトの整列条件を持ち，それらの型へのポインタ参照も 2 バイトの整列条件を持ちます。よって，パッキングしたメンバのポインタをパッキング対象でない型へのポインタに代入した場合，通常は動作を保証しません。本オプションを指定することでポインタの間接参照が 1 バイトアクセスとなるため，パッキングした構造体のメンバをポインタで間接参照できるようになります。

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。
- ccrl コマンドに対して，-prn_path，-mirror_source，-mirror_region，-define，-undefine，-include，-base_number，-convert_asm，-warning，または -no_warning オプションを指定する場合は，-asmopt オプションを使用する必要があります。
  また，-include オプションは，ccrl コマンドに対しては -I オプションとして指定することもできます。

オプションの分類と説明を以下に示します。

表 2.6 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	asrl のバージョン情報を表示します。
	-help	asrl のオプションの説明を表示します。
出力ファイル指定	-output	出力ファイル名を指定します。
	-obj_path	アセンブル後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。
	-prn_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。
ソース・デバッグ制御	-debug	ソース・デバッグ用の情報を出力します。
デバイス指定制御	-dev	対象デバイス・ファイルをパス付きで指定します。
	-cpu	CPU コアの種別を指定します。
	-mirror_source	MAA レジスタに設定する値を指定します。
	-mirror_region	ミラー先領域のアドレス範囲を指定します。
最適化	-goptimize	モジュール間最適化用の情報を生成します。
シンボル定義指定	-define	アセンブラ・シンボルを定義します。
	-undefine	-define 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。
インクルード・ファイル読み込みパス指定	-include	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。
入力ファイル制御	-character_set	日本語／中国語の文字コードを指定します。
	-base_number	数値定数の基数表現形式を指定します。
アセンブラ移行支援機能	-convert_asm	アセンブラ移行支援機能を有効にします。
エラー・メッセージ・ファイル出力指定	-error_file	エラー・メッセージをファイルに出力します。

分類	オプション	説明
警告メッセージ出力制御	-warning	指定した警告メッセージを出力します。
	-no_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
- -help

-V

asrl のバージョン情報を表示します。

[指定形式]

```
-V
```

- 省略時解釈
  asrl 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]

- asrl 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]

- asrl のバージョン情報を標準エラー出力に出力します。

```
>asrl -V
```

-help

asrl のオプションの説明を表示します。

[指定形式]

```
-help
```

- 省略時解釈
  asrl のオプションの説明を表示しません。

[詳細説明]

- asrl 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]

- asrl のオプションの説明を標準エラー出力に出力します。

```
>asrl -help
```

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。

- -output
- -obj_path
- -prn_path

-output

出力ファイル名を指定します。

[指定形式]

```
-output=file
```

- 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，アセンブル処理を継続できない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。

```
>asrl -output=sample.obj -dev=dr5f100pj.dvf main.asm
```

-obj_path

アセンブル後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]

```
-obj_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]

- アセンブル後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。

```
>asrl -obj_path=D:¥sample -dev=dr5f100pj.dvf main.asm
```

-prn_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]

```
-prn_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* にアセンブル・リスト・ファイルを保存します。
  入力ファイルの拡張子が “.asm”，“.s”，または “.fsy” の場合は，拡張子を “.prn” で置き換えたファイル名とします。
  それ以外の拡張子の場合は，拡張子を “.prn” を追加したファイル名とします。
  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・フォルダにアセンブル・リスト・ファイルを保存します。
  入力ファイルの拡張子が “.asm”，“.s”，または “.fsy” の場合は，拡張子を “.prn” で置き換えたファイル名とします。
  それ以外の拡張子の場合は，拡張子を “.prn” を追加したファイル名とします。

[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。

```
>asrl -prn_path=D:¥sample -dev=dr5f100pj.dvf 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。

- -debug

-debug

ソース・デバッグ用の情報を出力します。

[指定形式]

```
-debug
```

- 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。

```
>asrl -debug -dev=dr5f100pj.dvf main.asm
```

デバイス指定制御

デバイス指定制御オプションには，次のものがあります。

- -dev
- -cpu
- -mirror_source
- -mirror_region

-dev

対象デバイス・ファイルをパス付きで指定します。

[指定形式]

```
-dev=[path¥]file
```

- 省略時解釈
  -cpu オプションを指定している場合は，-cpu オプションによるコア種別の指定を有効にします。
  -cpu オプションを指定していない場合は，エラーとなります。

[詳細説明]

- 対象デバイス・ファイル *file* をパス付き（*path*）で指定します。
- 指定したデバイス・ファイルから読み込んだ情報を使用して，その情報に対応したオブジェクト・ファイルを生成します。
- 指定したデバイス・ファイルが見つからない場合は，エラーとなります。
- -cpu オプションと同時に指定した際，本オプションで指定したデバイス・ファイル中の CPU コア種別と -cpu オプションで指定した CPU コアが異なる場合は，エラーとなります。

[使用例]

- デバイス・ファイル DR5F100PJ.DVF を指定します。

```
>asrl -dev=dr5f100pj.dvf main.asm
```

-cpu

CPU コアの種別を指定します。

[指定形式]

```
-cpu={S1|S2|S3}
  S1     : RL78-S1 コア
  S2     : RL78-S2 コア
  S3     : RL78-S3 コア
```

- 省略時解釈
  -dev オプションを指定している場合は，-dev オプションで指定したデバイス・ファイルに書かれた CPU コアを有効にします。-dev オプションを指定していない場合は，エラーとなります。

[詳細説明]

- 各 CPU コアの共通オブジェクト・ファイルを出力します。
- 指定可能な文字列以外を指定した場合は，エラーとなります。
- -dev オプションと同時に指定した際，本オプションで指定した CPU コア種別と -dev オプションで指定したデバイス・ファイル中の CPU コアが異なる場合は，エラーとなります。
  それ以外の場合は，デバイス・ファイルを使用して処理を行います。
- -dev，および -cpu オプションを指定する際に同時に指定可能な他のオプションについて，以下に示します。

オプション	-dev オプション指定時	-cpu オプション指定時
-mirror_source	指定可能	指定可能[注]
-mirror_region	指定不可	指定可能

注 -cpu=S1 オプションと -mirror_source=1 オプションを同時に指定することはできません。

[使用例]

- CPU 種別を RL78-S2 コアに対応したコードを生成します。

```
>asrl -cpu=S2 main.c
```

-mirror_source

MAA レジスタに設定する値を指定します。

[指定形式]

```
-mirror_source={0|1|common}
```

- 省略時解釈
  -mirror_source=0 オプションの指定と同じです。

[詳細説明]

- MAA レジスタに設定する値を指定します。
- ミラー元セクションの配置先が 0x0xxxx 番地の場合は 0，0x1xxxx 番地の場合は 1 を指定します。
  本オプションは，絶対アドレス・セクションに所属するシンボルがミラー元領域にあるかどうかを判断するため，およびリンカにミラー元セクションの配置先を教えるために使用します。
  CPU コアの種別が RL78-S1 の場合は 0x0xxxx 番地固定のため指定自体が不要であり，1 を指定した場合はエラーとなります。
- common を指定した場合は，ミラー元領域に配置されたシンボルへの参照をサポートしません。また，ミラー元アドレスに対するミラー変換を行いません。
- -cpu=S1 オプション指定時に本オプションで 1 を指定した場合は，エラーとなります。
- -dev オプション指定時に本オプションを指定した場合は，内部 ROM（CodeFlash）のアドレス範囲とのチェックを行います。
- -mirror_source=common オプション指定時に -mirror_region オプションを指定した場合は，エラーとなります。

[使用例]

- ミラー元セクションの配置先に 0x0xxxx 番地を指定します。

```
>asrl -dev=dr5f100pj.dvf -mirror_source=0 main.asm
```

-mirror_region

ミラー先領域のアドレス範囲を指定します。

[指定形式]

```
-mirror_region=start_address,end_address
```

- 省略時解釈
  -dev オプションを指定していない場合，ミラー領域を持たないデバイスとみなして動作します。

[詳細説明]

- ミラー先領域のアドレス範囲（先頭アドレス，末尾アドレス）を指定します。
- ミラー元領域のアドレス範囲を算出するために使用します。
  CPU コアの種別が RL78-S1 の場合は 0xF8000 を，RL78-S2，または RL78-S3 の場合は 0xF0000 をこれらのアドレスから減算し，ミラー元領域のアドレス範囲とみなします。0xF0000 ～ 0xFFFFF の範囲外の値を指定した場合はエラーとなります。
- -dev オプション指定時に本オプションを指定した場合は，エラーとなります。
- -cpu オプション指定時に本オプションを指定しない場合は，ミラー領域を持たないデバイスとみなして動作します。
- -mirror_source=common オプション指定時に本オプションを指定した場合は，エラーとなります。

[使用例]

- ミラー先領域のアドレス範囲を指定します。

```
>asrl -cpu=S2 -mirror_region=0xf3000,0xfaeff main.asm
```

最適化

最適化オプションには，次のものがあります。

- -goptimize

-goptimize

モジュール間最適化用の情報を生成します。

[指定形式]

```
-goptimize
```

- 省略時解釈
  モジュール間最適化用の情報を生成しません。

[詳細説明]

- モジュール間最適化時に使用する付加情報を出力ファイル内部に生成します。
- 本オプションを指定したファイルは，リンク時にモジュール間最適化の対象になります。
  モジュール間最適化の詳細については，リンク・オプション -Optimize を参照してください。

[使用例]

- モジュール間最適化用の情報を生成します。

```
>asrl -goptimize -dev=dr5f100pj.dvf main.asm
```

シンボル定義指定

シンボル定義指定オプションには，次のものがあります。

- -define
- -undefine

-define

アセンブラのユーザ定義シンボル（ネーム）を定義します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- アセンブラのユーザ定義シンボル（ネーム）として *name* を定義します。
- *def* の指定方式は次の通りです。
  - 整数値のみ指定可能
  - 整数値以外を指定した場合は 0 とみなす
  - 整数値は 10 進，prefix 方式の 8 進 (0...)，16 進記法 (0x...) が可能
  - 先頭の - 符号は指定可能だが，+ 符号は不可
  - 負数は 2 の補数に変換される
- アセンブリ・ソース・プログラムの先頭で，*name* .SET *def* 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。
- =*def* を省略した場合，*def* を 1 とみなします。
- 本オプションは，複数指定が可能です。
- 本オプションと -undefine 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。

```
>asrl -define=sample=256 -dev=dr5f100pj.dvf main.asm
```

-undefine

-define 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]

```
-undefine=name[,name]...
```

- 省略時解釈
  なし

[詳細説明]

- -define オプションによる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。
- *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* .EQU *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。
- 本オプションと -define 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]

- -define 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。

```
>asrl -define=test -dev=dr5f100pj.dvf main.asm -undefine=test
```

インクルード・ファイル読み込みパス指定

インクルード・ファイル読み込みパス指定オプションには，次のものがあります。

- -include

-include

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]

```
-include=path[,path]...
```

- 省略時解釈
  なし

[詳細説明]

- $INCLUDE 制御命令，および $BINCLUDE 制御命令でインクルード・ファイル名をパスなし，または相対パスで指定した場合に，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> 本オプションで指定したフォルダ（指定が複数ある場合は，コマンド・ラインで指定した順（左から右の順））
  <2> $INCLUDE 制御命令，または $BINCLUDE 制御命令が記述されたソース・ファイルのあるフォルダ
  <3> カレント・フォルダ（asrl を起動したフォルダ）

- *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。

```
>asrl -include=D:¥include -dev=dr5f100pj.dvf D:¥src¥main.asm
```

入力ファイル制御

入力ファイル制御オプションには，次のものがあります。

- -character_set
- -base_number

-character_set

日本語／中国語の文字コードを指定します。

[指定形式]

```
-character_set={none|sjis|euc_jp|utf8|big5|gb2312}
```

- 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]

- 入力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。
- パラメータ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，入力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，動作は保証しません。

none	日本語／中国語の文字コードを処理しません
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

- パラメータを省略した場合は，エラーとなります。

[使用例]

- 入力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。

```
>asrl -character_set=euc_jp -dev=dr5f100pj.dvf main.asm
```

-base_number

数値定数の基数表現形式を指定します。

[指定形式]

```
-base_number={prefix|suffix}
```

- 省略時解釈
  -base_number=prefix オプションの指定と同じです。

[詳細説明]

- 数値定数の基数表現形式を指定します。
- パラメータ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。

prefix	Prefix 表現形式（0xn...n）を指定します。
suffix	Suffix 表現形式（n...nH）を指定します。

- パラメータを省略した場合は，エラーとなります。

[使用例]

- 数値定数の基数表現形式に Suffix 表現形式を指定します。

```
>asrl -base_number=suffix -dev=dr5f100pj.dvf main.asm
```

アセンブラ移行支援機能

アセンブラ移行支援機能オプションには，次のものがあります。

- -convert_asm

-convert_asm

アセンブラ移行支援機能を有効にします。

[指定形式]

```
-convert_asm
```

- 省略時解釈
  アセンブラ移行支援機能を有効にしません。

[詳細説明]

- 次の表中の「CA78K0R アセンブラ言語仕様」の記述を「RL78 アセンブラ言語仕様」に読み替えてアセンブルします。

表 2.7 アセンブラ移行支援機能

種別	CA78K0R アセンブラ言語仕様	RL78 アセンブラ言語仕様	ソース修正，オプション指定の必要性
数値定数	n…nB (n=0, 1) (2 進数)	同左	-base_number=suffix を指定してください。
	n…n0 (n=0 ～ 7) (8 進数)	同左	-base_number=suffix を指定してください。
	n…nH (n=0 ～ 9, A ～ F, a ～ f) (16 進数)	同左	-base_number=suffix を指定してください。
文字列	' 文字 ... 文字 '	" 文字 ... 文字 "	文字列中の 2 文字連続のシングルクオート (') を (¥') に変えて，文字全体をダブルクォート (") で囲んでください。 例 .「DB 'abc''de'」→「.DB "abc¥'de"」
オペランド欄	特殊機能レジスタ (SFR, 2ndSFR)	同左	-dev を指定してください。

種別	CA78K0R アセンブラ 言語仕様	RL78 アセンブラ 言語仕様	ソース修正，オプション指定の必要性
セグメント定義疑似命令	セグメントなし	.CSEG TEXT	
	CSEG 再配置属性なし	.CSEG TEXTF	
	CSEG CALLT0	.CSEG CALLT0	
	CSEG FIXED	.CSEG TEXT	
	CSEG BASE	.CSEG TEXT	
	CSEG AT	.CSEG AT	
	CSEG UNIT	.CSEG TEXTF	
	CSEG UNITP	.CSEG TEXTF	.ALIGN 2 を追加してください。
	CSEG IXRAM	.CSEG TEXTF	
	CSEG OPT_BYTE	.CSEG OPT_BYTE	
	CSEG SECUR_ID	.CSEG SECUR_ID	
	CSEG PAGE64KP	.CSEG TEXTF_UNIT64KP	
	CSEG UNIT64KP	.CSEG TEXTF_UNIT64KP	
	CSEG MIRRORP	.CSEG CONST	
	DSEG 再配置属性なし	.DSEG BSSF	
	DSEG SADDR	.DSEG SBSS	
	DSEG SADDRP	.DSEG SBSS	
	DSEG AT	.DSEG BSS_AT	
	DSEG UNIT	.DSEG BSS	
	DSEG UNITP	.DSEG BSS	
	DSEG IHRAM	.DSEG BSS	
	DSEG LRAM	.DSEG BSS	
	DSEG DSPRAM	.DSEG BSS	
	DSEG IXRAM	.DSEG BSS	
	DSEG BASEP	.DSEG BSS	
	DSEG PAGE64KP	.DSEG BSS	
	DSEG UNIT64KP	.DSEG BSS	
	BSEG 再配置属性なし	.BSEG SBSS_BIT	
	BSEG UNIT	.BSEG SBSS_BIT	
	BSEG AT	.BSEG BIT_AT	
	ORG	.ORG	
シンボル定義疑似命令	EQU	.EQU	オペランドにはリロケータブルなラベルを記述できません。
	SET	.SET	

種別	CA78K0R アセンブラ 言語仕様	RL78 アセンブラ 言語仕様	ソース修正，オプション指定の必要性
メモリ初期化，領域確保疑似命令	DB	.DB	サイズ指定は記述を変更してください。
	DW	.DB2	サイズ指定は記述を変更してください。オペランドが文字列定数の場合は，文字列に変えてください。 例．「DW 'ab'」→「.DB "ba"」
	DG	.DB4	サイズ指定は記述を変更してください。オペランドが文字列定数の場合は，文字列に変えてください。 例．「DG 'ab'」→「.DB "ba¥0¥0"」
	DS	.DS	
	DBIT	.DBIT	
リンケージ疑似命令	PUBLIC	.PUBLIC	
	EXTRN	.EXTERN	
	EXTBIT	.EXTBIT	
オブジェクト・モジュール名宣言疑似命令	NAME	コメント扱い	
分岐命令自動選択疑似命令	BR	BR !!addr20	
	CALL	CALL !!addr20	
アセンブル終了疑似命令	END	コメント扱い	END 以降の記述が有効になるので，無効にしてください。
アセンブル対象品種指定制御命令	$PROCESSOR($PC)	コメント扱い	-dev を指定してください。
デバッグ情報出力制御命令	$DEBUG($DG)	コメント扱い	-debug を指定してください。
	$NODEBUG($NODG)	コメント扱い	-debug を指定してください。
	$DEBUGA	コメント扱い	-debug を指定してください。
	$NODEBUGA	コメント扱い	-debug を指定してください。
クロスリファレンス・リスト出力指定制御命令	$XREF($XR)	コメント扱い	
	$NOXREF($NOXR)	コメント扱い	
	$SYMLIST	コメント扱い	
	$NOSYMLIST	コメント扱い	
インクルード制御命令	$INCLUDE($IC)	$INCLUDE	

種別	CA78K0R アセンブラ言語仕様	RL78 アセンブラ言語仕様	ソース修正，オプション指定の必要性
アセンブル・リスト制御命令	$EJECT($EJ)	コメント扱い	
	$LIST($LI)	コメント扱い	
	$NOLIST($NOLI)	コメント扱い	
	$GEN	コメント扱い	
	$NOGEN	コメント扱い	
	$COND	コメント扱い	
	$NOCOND	コメント扱い	
	$TITLE($TT)	コメント扱い	
	$SUBTITLE($ST)	コメント扱い	
	$FORMFEED	コメント扱い	
	$NOFORMFEED	コメント扱い	
	$WIDTH	コメント扱い	
	$LENGTH	コメント扱い	
	$TAB	コメント扱い	
条件付きアセンブル制御命令	$IF(スイッチ名)	同左	-define= スイッチ名 =1，または，-define= スイッチ名 =0 を指定してください。
	$IF(スイッチ名 : スイッチ名 ...)	$IF(スイッチ名 | スイッチ名 ...)	-define= スイッチ名 =1，または，-define= スイッチ名 =0 を指定してください。 あるいは， スイッチ名 .SET 1，または，スイッチ名 .SET 0 を追加してください。
	$_IF	$IF	
	$ELSEIF(スイッチ名 : スイッチ名 ...)	$ELSEIF(スイッチ名 | スイッチ名 ...)	-define= スイッチ名 =1，または，-define= スイッチ名 =0 を指定してください。 あるいは， スイッチ名 .SET 1，または，スイッチ名 .SET 0 を追加してください。
	$_ELSEIF	$ELSEIF	
	$SET	コメント扱い	
	$RESET	コメント扱い	
漢字コード制御命令	$KANJICODE	コメント扱い	-character_set を指定してください。
RAM 領域配置指定制御命令	$RAM_ALLOCATE	コメント扱い	対象セグメントを .CSEG TEXTF_UNIT64KP で配置してください。
その他の制御命令	$TOL_INF	コメント扱い	
	$DGS	コメント扱い	
	$DGL	コメント扱い	

［注意］

- 上記の表に記載していない CA78K0R アセンブラの言語仕様については，ソースを修正する必要があります。
- シンボル定義疑似命令 EQU のオペランドには，リロケータブルなラベルを記述できません。
  この場合，EQU の左辺にあるネームの参照個所をリロケータブルなラベルに置き換えて，EQU 自体は無効化してください。

  例 1.

```
DMAINP  DSEG    SADDRP
RABUF1: DS      8
RABUF2: DS      8
OFFSET  EQU     RABUF2 - RABUF1   ; E0551203: リロケータブル項は記述できません。
FPREAD  EQU     RABUF1.4          ; E0551203: リロケータブル項は記述できません。
        CSEG
        ADD     A,#OFFSET
        CLR1    FPREAD
        END
```

  (修正方法)

```
OFFSET  EQU     RABUF2 - RABUF1
FPREAD  EQU     RABUF1.4
```

  を無効にして，

```
        ADD     A,#OFFSET
        CLR1    FPREAD
```

  を

```
        ADD     A,#RABUF2 - RABUF1
        CLR1    RABUF1.4
```

  に修正します。

- アドレス幅が 16 ビットを超えると，リンク時にエラーとなります。
  この場合，アドレスに LOWW 演算子を付加してください。

  例 2.

```
DMAINP  DSEG    SADDRP
RABUF1: DS      8
        CSEG
        MOVW    HL,#RABUF1  ; E0562330:Relocation size overflow
        END
```

  (修正方法)

```
        MOVW    HL,#RABUF1
```

  を

```
        MOVW    HL,#LOWW RABUF1
```

  に修正します。

- メモリ初期化や領域確保疑似命令における「(サイズ)」オペランドの解釈が異なります。
  以下の例に従って修正してください。

  例 3.

```
        CSEG
        DW      (3)
        END
```

(修正方法)

```
        .CSEG
        .DS     6
        .END
```

CA78K0R アセンブラの場合：3 つのワード領域 (6 バイト) を 00H で初期化します。
RL78 アセンブラの場合：1 ワードの領域 (2 バイト) を 3 で初期化します。

- RL78 アセンブラで許していない演算を別の方法に置き換えてください。

例 4.

```
MSGDATA CSEG    AT 80H
TMSGOK:
        DB      'OK'
        CSEG
        MOV     H,#LOWW TMSGOK/100H     ; E0551215: ラベルの記述に誤りがあります。
                                        ; E0550250:100H の構成に誤りがあります。
        END
```

(修正方法)

```
        MOV     H,#LOWW TMSGOK/100H
```

を

```
        MOV     H,#HIGH TMSGOK
```

に修正します。

- セグメント定義疑似命令 CSEG UNITP を .CSEG TEXTF + .ALIGN 2 に変えてください。

例 5.

```
XMAIN2  CSEG    UNITP
TINTVL: DW      47999
        END
```

(修正方法)

```
XMAIN2  CSEG    UNITP
```

を

```
XMAIN2  .CSEG   TEXTF
        .ALIGN  2
```

に修正します。

[使用例]

- アセンブラ移行支援機能を有効にします。

```
>asrl -convert_asm -cpu=S2 -dev=dr5f100pj.dvf main.asm
```

エラー・メッセージ・ファイル出力指定

エラー・メッセージ・ファイル出力指定オプションには，次のものがあります。

- -error_file

-error_file

エラー・メッセージをファイルに出力します。

[指定形式]

```
-error_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。

```
>asrl -error_file=err -dev=dr5f100pj.dvf 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。

- -warning
- -no_warning

-warning

指定した警告メッセージを出力します。

[指定形式]

```
-warning={num|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を出力します。
- *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。
- 本オプションで制御することができるのは，メッセージ番号（コンポーネント番号を含む）が 0550000 ～ 0559999 であるもののみです。

[使用例]

- 警告メッセージ W0550001，W0550005 を出力します。

```
>asrl -warning=50001,50005 -dev=dr5f100pj.dvf main.asm
```

-no_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]

```
-no_warning={num|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。
- 本オプションで制御することができるのは，メッセージ番号（コンポーネント番号を含む）が 0550000 ～ 0559999 であるもののみです。

[使用例]

- 警告メッセージ W0550001，W0550005 の出力を抑止します。

```
>asrl -no_warning=50001,50005 -dev=dr5f100pj.dvf 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。

```
>asrl @command.txt main.asm
```

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。
- ccrl コマンドに対して，リンク・オプションを指定する場合は，-lnkopt 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 2.8 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。
	-ALLOW_DUPLICATE_MODULE_NAME 【V1.09 以降】	複数の同じモジュール名の指定を許可します。

分類	オプション	説明
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。
	-END_RECORD 【V1.05 以降】	エンドレコードを指定します。
	-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。
	-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。
	-FIX_RECORD_LENGTH_AND_ALIGN【V1.06 以降】	データ・レコードの出力フォーマットを固定します。
	-PADDING	セクションの終端にデータを埋め込みます。
	-CRc	CRC 演算を行うかどうかを指定します。
	-VECT	ベクタテーブル・アドレスの空き領域にアドレスを設定します。
	-VECTN	特定ベクタテーブル・アドレスの領域にアドレスを設定します。
	-SPLIT_VECT 【V1.07 以降】	ベクタテーブル・セクションを分割して生成します。
	-VFINFO	変数 / 関数情報ファイルを出力します。
	-CFI【Professional 版のみ】【V1.06 以降】	不正な間接関数呼び出し検出で用いる関数リストを生成します。
	-CFI_ADD_Func【Professional 版のみ】【V1.06 以降】	不正な間接関数呼び出し検出で用いる関数リストに追加する関数シンボルまたはアドレスを指定します。
	-CFI_IGNORE_Module【Professional 版のみ】【V1.06 以降】	不正な間接関数呼び出し検出で用いる関数リストから除外するモジュールを指定します。
	-RAM_INIT_TABLE_SECTION【V1.12 以降】	RAM 初期化のための情報テーブルを生成します。
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。
最適化	-Optimize	モジュール間最適化の実行有無を指定します。
	-NOOptimize	モジュール間最適化を行いません。
	-SEction_forbid	特定セクションの最適化を抑止します。
	-Absolute_forbid	アドレス＋サイズの範囲の最適化を抑止します。
	-SYmbol_forbid 【V1.02 以降】	未参照シンボル削除の抑止シンボルを指定します。
	-ALLOW_OPTIMIZE_ENTRY_BLOCK【V1.13 以降】	実行開始シンボルより前に配置されている領域を最適化の対象にします。

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。
	-USER_OPT_BYTE	ユーザ・オプション・バイトに設定する値を指定します。
	-OCDBG	オンチップ・デバッグ制御値に設定する値を指定します。
	-SECURITY_OPT_BYTE【V1.12 以降】	セキュリティ・オプション・バイト制御値に設定する値を指定します。
	-SECURITY_ID	セキュリティ ID 値を指定します。
	-FLASH_SECURITY_ID【V1.12 以降】	フラッシュ・プログラマ・セキュリティ ID 値に設定する値を指定します。
	-AUTO_SECTION_LAYOUT	セクションを自動配置します。
	-SPLIT_SECTION【V1.12 以降】	セクションの自動配置で配置単位をモジュール別のセクション単位で行います。
	-STRIDE_DSP_MEMORY_AREA【V1.12 以降】	自動配置のセクション割り付けは，FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域で分断した領域に分けて配置します。
	-DEBUG_MONITOR	OCD モニタ領域を指定します。
	-RRM	RRM/DMM 機能用ワーク領域を指定します。
	-SELF	セルフ RAM 領域にセクションを配置しません。
	-SELFW	セルフ RAM 領域にセクションを配置した場合は警告を出力します。
	-OCDTR	トレース RAM とセルフ RAM 領域にセクションを配置しません。
	-OCDTRW	トレース RAM とセルフ RAM 領域にセクションを配置した場合は警告を出力します。
	-OCDHPI	ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置しません。
	-OCDHPIW	ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置した場合は警告を出力します。
	-DSP_MEMORY_AREA【V1.12 以降】	FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域にセクションを配置しません。
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。
	-CHECK_DEVICE	オブジェクト・ファイルの作成時に指定したデバイス・ファイルのチェックを行います。
	-CHECK_64K_ONLY	セクションが（64K-1）バイト境界をまたいで配置されるかをチェックしません。
	-NO_CHECK_SECTION_LAYOUT	セクションの割り付けアドレスがデバイス・ファイルの情報と整合するかをチェックしません。
	-CHECK_OUTPUT_ROM_AREA【V1.07 以降】	ヘキサ・ファイルの出力アドレス範囲が，内部 ROM またはデータフラッシュにあることをチェックします。
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。
マイコン指定	-DEVICE	デバイス・ファイル名を指定します。

分類	オプション	説明
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。
	-LIB_REName 【V1.08 以降】	ライブラリから入力されたシンボルやセクションの名前を変更します。
	-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。
	-VERBOSE 【V1.10 以降】	詳細情報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
- -ALLOW_DUPLICATE_MODULE_NAME 【V1.09 以降】

-Input

入力ファイルを指定します。

[指定形式]

```
-Input=suboption[{,| △} ...]
  suboption := {file|file(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=suboption[, ...]
  suboption := 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。
  【V1.04 以前】
- セクション属性 *attribute* に指定可能なものは，CALLT0，CODE，TEXT，TEXTF，TEXTF_UNIT64KP，CONST，CONSTF，SDATA，DATA，DATAF，OPT_BYTE，または SECUR_ID です。CODE は TEXT の再配置属性と同じになります。OPT_BYTE を指定してセクション名に “.option_byte” 以外を指示した場合はエラーとなります。
  SECUR_ID を指定してセクション名に “.security_id” 以外を指示した場合はエラーとなります。
  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。
  【V1.05 以降】
- 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。

[制限]

- 本オプションで指定したバイナリ・ファイルは，0 ～ 0x0FFFF 番地にしか配置できません。【V1.04 以前】
  0x10000 番地以降にバイナリ・ファイルを配置するなど，希望するセクション属性に変更したい場合は，下記のようなアセンブラソースを作成してください。

```
    .SECTION BIN_SEC, TEXTF
    $BINCLUDE(tp.bin)
```

[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=suboption[, ...]
  suboption := {symbol1=symbol2|symbol1=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。
- コンパイル時に entry シンボルを指定している場合は，本オプションの指定を優先します。

[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。
- モジュール間最適化（-optimize[={symbol_delete|speed}]）指定時には，本オプションで *symbol* を指定する必要があります。指定がない場合は，モジュール間最適化指定は無効となります。また，本オプションで *address* を指定している場合は，未参照シンボル削除最適化を抑止します。
- -start オプションで配置アドレスを指定したセクションのリスト中に，-entry で指定したアドレスが所属するとき，その -start オプションで指定した配置アドレスから，-entry で指定したアドレスまでの領域を対象にした最適化は抑止します。

-ALLOW_DUPLICATE_MODULE_NAME 【V1.09 以降】

複数の同じモジュール名からの，ライブラリ生成を許可します。

[指定形式]

```
-ALLOW_DUPLICATE_MODULE_NAME
```

- 省略時解釈
  なし

[詳細説明]

- ライブラリ生成時に，複数の同じモジュール名の入力ファイル指定を許容します。
- ライブラリ内に既に名前が重複するモジュールがあれば，モジュール名の末尾に ".<N>" を加えてライブラリに登録します。
- <N> にはライブラリ中で重複しないモジュール名になるよう番号を設定します。重複しない番号を見つけられない場合はエラーを出力して終了します。

[使用例]

- ライブラリ（a.lib）を同じモジュール名（mod）を持つ複数の入力ファイルから生成します。

```
> rlink -allow_duplicate_module_name -form=lib -output=a.lib b¥mod.obj c¥mod.obj
d¥mod.obj
```

生成したライブラリ（a.lib）は次のように構成されます。

- mod（b¥mod.obj から）
- mod.1（c¥mod.obj から）
- mod.2（d¥mod.obj から）

[備考]

- 本オプションは，-form={object|absolute|relocate|hexadecimal|stype|binary} オプション，-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。

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
- -END_RECORD 【V1.05 以降】
- -ROm
- -OUtput
- -SPace
- -Message
- -NOMessage
- -MSg_unused
- -BYte_count
- -FIX_RECORD_LENGTH_AND_ALIGN【V1.06 以降】
- -PADDING
- -CRc
- -VECT
- -VECTN
- -SPLIT_VECT 【V1.07 以降】
- -VFINFO
- -CFI【Professional 版のみ】【V1.06 以降】
- -CFI_ADD_Func【Professional 版のみ】【V1.06 以降】
- -CFI_IGNORE_Module【Professional 版のみ】【V1.06 以降】
- -RAM_INIT_TABLE_SECTION【V1.12 以降】

-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。

[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。

表 2.9 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，-hide，-optimize， -nooptimize，-symbol_forbid，-section_forbid， -absolute_forbid，-compress，-nocompress，-rename， -lib_rename，-delete，-define，-fsymbol，-stack， -memory，-msg_unused， -show={symbol|reference|xreference|total_size|vector| struct|relocation_attribute|cfi|all}， -user_opt_byte，-ocdbg，-security_id，-device， -padding，-vect，-vectn，-split_vect，-vfinfo， -auto_section_layout，-debug_monitor，-rrm，-self， -selfw，-ocdtr，-ocdtrw，-ocdhpi，-ocdhpiw， -check_device，-check_64k_only， -no_check_section_layout，-cfi，-cfi_add_func， -cfi_ignore_module
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，-lib_rename，-delete， -show={symbol|xreference|total_size|all}， -device，-check_device
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，-space，-optimize，-nooptimize， -symbol_forbid，-section_forbid，-absolute_forbid， -rename，-lib_rename，-delete，-define，-fsymbol， -stack，-record，-end_record[注 2]，-s9[注 2]， -byte_count[注 3]，-fix_record_length_and_align[注 6]， -memory，-msg_unused， -show={symbol|reference|xreference|total_size|vector| struct|relocation_attribute|cfi|all}， -user_opt_byte，-ocdbg，-security_id，-crc，-device， -padding，-vect，-vectn，-split_vect，-vfinfo， -auto_section_layout，-debug_monitor，-rrm，-self， -selfw，-ocdtr，-ocdtrw，-ocdhpi，-ocdhpiw， -check_device，-check_64k_only， -no_check_section_layout， -check_output_rom_area[注 7]，-cfi，-cfi_add_func， -cfi_ignore_module
		ロード・モジュール・ファイル	-input，-output，-record，-end_record[注 2]，-s9[注 2]， -byte_count[注 3]，-fix_record_length_and_align[注 6]， -show={symbol|reference|xreference|all}， -device，-check_output_rom_area[注 7]
		インテル拡張ヘキサ・ファイル[注 4]	-input，-output -device，-check_output_rom_area[注 7]
		モトローラ・S タイプ・ファイル[注 4]	-input，-output，-s9[注 2]， -device，-check_output_rom_area[注 7]

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
Library	-strip あり	ライブラリ・ファイル	-library，-output，-memory 注 5
	-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 5，-show={symbol|section|all}，-allow_duplicate_module_name

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. -end_record および -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，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 4. 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 5. -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。

注 6. -fix_record_length_and_align 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 7. -check_output_rom_area 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。

［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。

-END_RECORD 【V1.05 以降】

エンドレコードを指定します。

[指定形式]

```
-END_RECORD=record
```

- 省略時解釈
  エントリ・ポイント・アドレスに合わせてエンドレコードを出力します。

[詳細説明]

- モトローラ・S タイプ・ファイルのエンドレコードを指定します。
- *record* に指定可能なものを以下に示します。

S7	S7 レコード
S8	S8 レコード
S9	S9 レコード

- 指定したアドレスフィールドより大きいエントリ・ポイント・アドレスの場合，エントリ・ポイント・アドレスに合わせてエンド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・S タイプ・エンド・レコードで出力します。

```
> rlink a.obj b.obj -end_record=S7 -form=stype -output=c.mot
```

[備考]

- -form={stype} 指定がない時，本オプションはエラーを出力して終了します。

-ROm

ROM から RAM へマップするセクションを指定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- 初期化データ領域の ROM 用，RAM 用領域を確保し，ROM セクション内定義シンボルを RAM セクション内アドレス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。
- *ROMsection*，*RAMsection* には，ワイルドカード記号 '*' を使用することができます。【V1.13 以降】
  - 初期値のあるリロケータブル・セクション（ROM 側セクション）の名前が，*ROMsection* のワイルドカード表現と一致する場合，*RAMsection* 内のワイルドカード記号 '*' の部分を，ROM 側セクション名の中のワイルドカード記号 '*' に一致する部分と置き換えて，RAM 側セクションの名前として処理します。

    例） ROM 側セクションが .data，.data_1，.sdata，.sdata_1 の 4 つあるとき，-rom=*data*=*data*_R と指定すると，.data_R，.data_1_R，.sdata_R，.sdata_1_R の 4 つの RAM 側セクションが生成されます。

    注 置換後の RAM 側セクション名は，-start オプションなどで適切に対応する必要があります。
  - ワイルドカード記号 '*' は複数個指定できますが，*ROMsection*，*RAMsection* で個数が一致している必要があります。

    例）

```
-rom=.data*=.data*_R      # 問題なし
-rom=.data*=.data*_R_*    # RAMsection 側のワイルドカード記号が多すぎるのでエラー
```

  - 置換によってできたセクション名と同名のセクションがすでにあった場合は，エラーになります。

[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/FAF00
```

[備考]

- 本オプションは，-form={object|relocate|library} オプション，-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。

-OUtput

出力ファイルを指定します。

[指定形式]

```
-OUtput=suboption[, ...]
【V1.06 以前】
  suboption := {file|file=range}
  range := {address1-address2|section[: ...]}
【V1.07 以降】
  suboption := {file|file=range|file=range/load-address|file=/load-address}
  range := {address1-address2|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。
- *load-address* を指定すると，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・ファイルを出力する際，ファイル上の先頭ロード・アドレスを *load-address* で指定した値に変更します。

[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）。
- *load-address* は，-form={hexadecimal | stype} オプションを指定した場合のみ指定可能です。

-SPace

出力範囲における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，出力範囲の指定方法によって，以下のように異なります。
  - -Output オプションの出力範囲がセクション指定の場合
    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。
  - -Output オプションの出力範囲がアドレス範囲指定の場合
    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。
  - -FIX_RECORD_LENGTH_AND_ALIGN オプションを指定した場合
    セクション先頭アドレスをアライメント数で割り切れるアドレスに整合したとき，セクション先頭に空き容量が存在する場合に指定データを出力します。
    オプションで指定したデータ・レコード長にセクション終端が満たない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]

- 0x100 番地から 0x2FF 番地の範囲内で，メモリの空き領域に対して 0xff で充てんします。

```
>rlink a.obj b.obj -form=hexadecimal -output=file1=100-2ff -start=.SEC1/100,.SEC2/
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，かつ -fix_record_length_and_align オプション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・レベルの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・レベルの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・レベルのメッセージを出力します。

[使用例]

- インフォメーション・レベルの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}[, ...]]
```

- 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出力します。

[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，検出するために使用します。

[使用例]

- 参照されない外部定義シンボルを出力します。

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

[備考]

- 本オプションは，-form={object|relocate|library} 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。
- 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。
- 以下のどちらかに該当する場合，参照関係の解析が正しく行うことができず，メッセージ出力により通知される情報が不正確となります。
  - 同一ファイル内の定数シンボルへの参照がある場合
  - コンパイル時に最適化が有効で，直下の関数を呼び出す場合

-BYte_count

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
  -form=hexadecimal 指定時は，バイト数の最大値を 0xFF として，インテル拡張ヘキサ・ファイルを生成します。
  -form=stype 指定時は，バイト数の最大値を 0x10 として，モトローラ・S タイプ・ファイルを生成します。

[詳細説明]

- インテル拡張ヘキサ・ファイルまたはモトローラ・S タイプ・ファイルを生成する際に，データ・レコード長を指定するためのオプションです。
- インテル拡張ヘキサ・ファイルを生成する場合，数値には，01 ～ FF（16 進数）を指定することができます。
- モトローラ・S タイプ・ファイルを生成する場合，数値には，次のいずれかの値を指定することができます。
  - S1 形式： 01 ～ FC（16 進数）
  - S2 形式： 01 ～ FB（16 進数）
  - S3 形式： 01 ～ FA（16 進数）

[使用例]

- データ・レコードのバイト数の最大値として 0x10 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，-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。

-FIX_RECORD_LENGTH_AND_ALIGN【V1.06 以降】

データ・レコードの出力フォーマットを固定します。

[指定形式]

```
-FIX_RECORD_LENGTH_AND_ALIGN=align
```

- 省略時解釈
  なし

[詳細説明]

- 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・ファイルを生成する際に，指定したアライメント数で整合したアドレスから常に一定数のレコード長で出力します。
- 出力開始アドレスは，セクション先頭アドレス以降，指定したアライメント数で割り切れる最初のアドレスとなります。
- レコード長は，-BYte_count オプションで指定した個数または規定値を常に出力します。
- レコード長を一定にすることより，一つのレコードに複数のセクションが出力される場合があります。
- 空きエリアには，-SPace オプションで指定した値か，-SPace オプションの指定がなく -CRc オプションの指定がある場合は 0xff，-SPace オプションの指定も -CRc オプションの指定もない場合は 0x00 を出力します。

[使用例]

- データレコードの出力を 8 で割り切れるアドレスから開始し，レコード長を 16 バイトに固定します。

```
>rlink a.obj b.obj -form=hexadecimal -byte_count=10 -fix_record_length_and_align=8
```

[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。

-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。
- 命令，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・データを埋め込みます。初期値がない変数のセクションは対象としません。

[使用例]

-以下の場合，.SEC1 セクションに 1 バイトのパディング・データを埋め込み，サイズを 0x06 バイトにしてリンク処理を行います。
  .SEC1 セクションのアライメント ： 2 バイト
  .SEC1 セクションのサイズ ： 0x05 バイト
  .SEC2 セクションのアライメント ： 1 バイト
  .SEC2 セクションのサイズ ： 0x03 バイト

```
>rlink a.obj b.obj -start=.SEC1,.SEC2/0 -padding
```

- 以下の場合，.SEC1 セクションに 1 バイトのパディング・データを埋め込み，サイズを 0x06 バイトにしてリンク処理を行うと，.SEC2 セクションと重複してしまうため，エラーを出力します。
  .SEC1 セクションのアライメント ： 2 バイト
  .SEC1 セクションのサイズ ： 0x05 バイト
  .SEC2 セクションのアライメント ： 1 バイト
  .SEC2 セクションのサイズ ： 0x03 バイト

```
>rlink a.obj b.obj -start=.SEC1/0,.SEC2/5 -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。

-CRc

CRC 演算を行うかどうかを指定します。

［指定形式］

```
-CRc=address=operation-range[/operation-method][(initial-value)][:endian]
    operation-range  := start-end[,start-end]... 【V1.01】
                        {start-end|section}[,{start-end|section}]... 【V1.02 以降】
    operation-method := {16-CCITT-MSB-LITTLE-4|16-CCITT-LSB|SENT-MSB} 【V1.01】
                        {CCITT|16-CCITT-MSB|16-CCITT-MSB-LITTLE-4|16-CCITT-MSB-
                         LITTLE-2|16-CCITT-LSB|16|SENT-MSB|32-ETHERNET} 【V1.02 以降】
    endian           := {BIG|LITTLE}[-size-offset]
```

- 省略時解釈
  CRC 演算，および演算結果の出力を行いません。

［詳細説明］

- 指定した範囲のロード・オブジェクトを，下位アドレスから上位アドレスの順で CRC（Cyclic Redundancy Check）演算を行い，演算結果を出力アドレスへエンディアンの指定方式で出力します。
- *address* には出力アドレスを指定します。
  指定可能な値の範囲は，0x0 ～ 0xFFFFF です。
- *start*，*end* には演算範囲（開始アドレス，終了アドレス）を指定します。
  指定可能な値の範囲は，0x0 ～ 0xFFFFF です。
- 演算方法には以下のいずれかを指定します。

演算方法	内容
CCITT 【V1.02 以降】	CRC-16-CCITT で MSB First，初期値 0xFFFF，XOR 反転による演算結果を得ることができます。 生成多項式は $x^{16}+x^{12}+x^5+1$ です。
16-CCITT-MSB 【V1.02 以降】	CRC-16-CCITT で MSB First による演算結果を得ることができます。 生成多項式は $x^{16}+x^{12}+x^5+1$ です。
16-CCITT-MSB-LITTLE-4	入力を LITTLE エンディアン 4 バイト単位とし CRC-16-CCITT で MSB First による演算結果を得ることができます。 生成多項式は $x^{16}+x^{12}+x^5+1$ です。
16-CCITT-MSB-LITTLE-2 【V1.02 以降】	入力を LITTLE エンディアン 2 バイト単位とし CRC-16-CCITT で MSB First による演算結果を得ることができます。 生成多項式は $x^{16}+x^{12}+x^5+1$ です。
16-CCITT-LSB	CRC-16-CCITT で LSB First による演算結果を得ることができます。 生成多項式は $x^{16}+x^{12}+x^5+1$ です。
16 【V1.02 以降】	CRC-16 で LSB First による演算結果を得ることができます。 生成多項式は $x^{16}+x^{15}+x^2+1$ です。
SENT-MSB	入力を LITTLE エンディアン 1 バイト中下位 4bit 単位とし SENT 準拠で初期値 0x5，MSB First による演算結果を得ることができます。 生成多項式は $x^4+x^3+x^2+1$ です。
32-ETHERNET 【V1.02 以降】	CRC-32-ETHERNET による演算結果を得ることができます。演算結果は初期値 0xFFFFFFFF，XOR 反転，ビットリバースされています。 生成多項式は $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$ です。

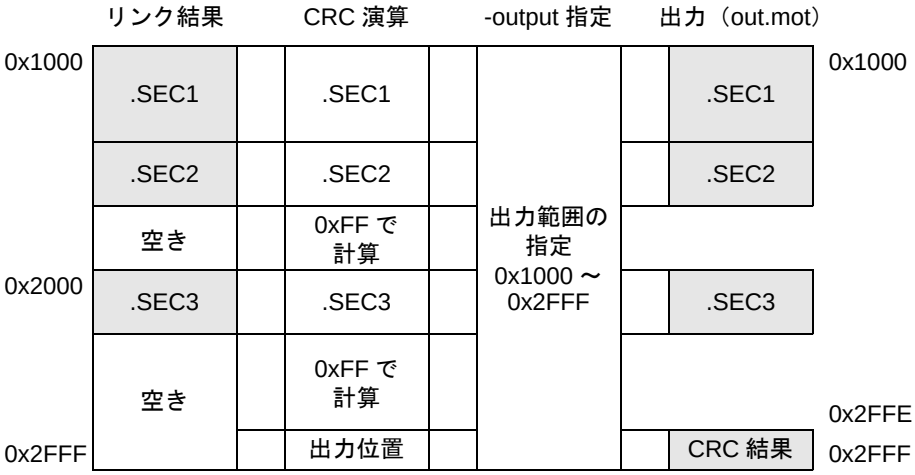
なお，演算方法の指定を省略した場合は，16-CCITT-MSB-LITTLE-4 を指定したものとします。

- *initial-value* には演算用の初期値を指定します。
  指定可能な値の範囲は，演算方法が 32-ETHERNET の場合は 0x0 ～ 0xFFFFFFFF，SENT-MSB では 0x0 ～ 0xF，それ以外は 0x0 ～ 0xFFFF です。
  初期値を省略した場合は，演算方法が SENT-MSB の場合は 0x5，CCITT の場合は 0xFFFF，32-ETHERNET の場合は 0xFFFFFFFF，それ以外は 0x0 を指定したものとして演算を行います。
- *endian* にはエンディアン，サイズ，オフセットを指定します。
  指定可能な組み合わせを以下に示します。
  - LITTLE
  - LITTLE-2-0
  - LITTLE-4-0
  - BIG
  - BIG-2-0
  - BIG-4-0
- エンディアンの BIG または LITTLE 指定が省略された場合は，入力オブジェクト・ファイルのエンディアンと同じになります。
- 演算結果の出力アドレスへの出力は，サイズで確保した領域の先頭からオフセットの位置に，BIG か LITTLE で指定したバイト・オーダーで書き込みます。確保した領域の先頭からオフセットの位置直前までは 0 を出力します。
- サイズとオフセットを省略した場合，サイズは 2 バイト，オフセットは 0 とします。
- 演算範囲にある空き領域は，-space オプションを指定していない場合は -space=FF オプションを指定していると仮定して，CRC 演算を行います。
  ただし，CRC 演算は，空き領域では 0xFF で計算を行いますが，0xFF を埋めることはありません。
  演算範囲として指定した下位アドレスから上位アドレスの順に演算を行います。
- 本オプションを複数回指定した場合，指定した全ての CRC 演算の結果を出力します。【V1.12 以降】

［使用例］

例 1.

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



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

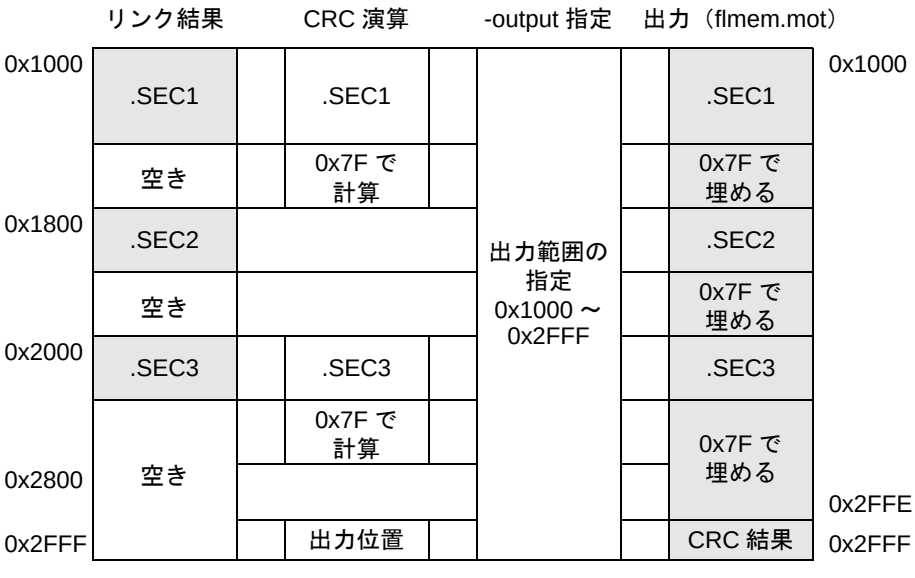
-output オプション：-output=out.mot=1000-2FFF
-space オプションを指定していないため，空き領域は out.mot に出力されません。
CRC 演算は，空き領域では 0xFF で計算を行いますが，0xFF を埋めることはありません。

注意 1. CRC 出力位置は計算範囲に含むことはできません。

注意 2. CRC 出力位置は -output オプションの出力範囲に含まれている必要があります。

例 2.

```
>rlink *.obj -form=stype -start=.SEC1/1000,.SEC2/1800,.SEC3/2000 -space=7F
-crc=2FFE=1000-17FF,2000-27FF -output=out.mot=1000-2FFF
```



-crc オプション：-crc=2FFE=1000-17FD,2000-27FF
0x1000 ～ 0x17FD と 0x2000 ～ 0x27FF の 2 つの領域に対して CRC 演算を行い，その結果を 0x2FFE 番地に出力します。
CRC 演算は，計算対象として連続していない複数の計算範囲を指定することができます。

-space オプション：-space=7F
指定した計算範囲の空き領域は -space オプションの値（0x7F）で計算されます。

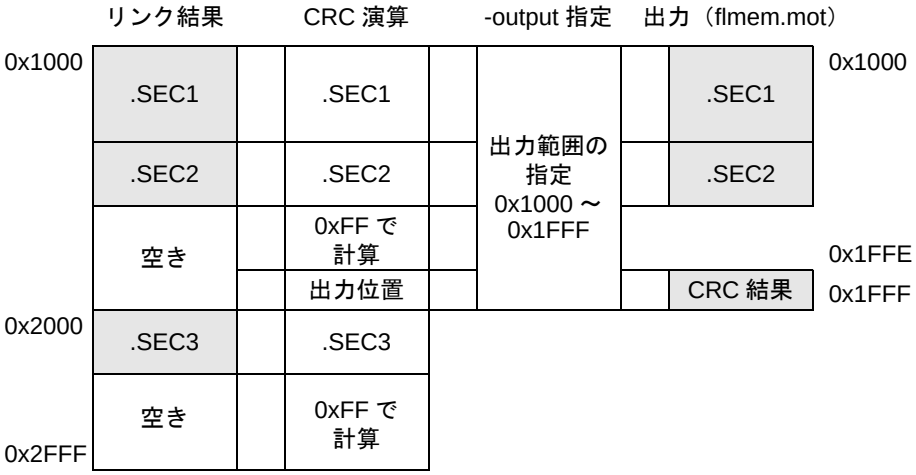
-output オプション：-output=out.mot=1000-2FFF
-space オプションを指定しているため，空き領域は out.mot に出力されます。
空き領域は 0x7F で充填されます。

注意 1. CRC 演算の計算順は計算範囲の指定順ではありません。下位アドレスから上位アドレスの順に計算されます。

注意 2. -crc オプションと -space オプションを同時に指定する場合，-space オプションに random，または 2 バイト以上の値を指定することはできません。1 バイトのデータを指定してください。

例 3.

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

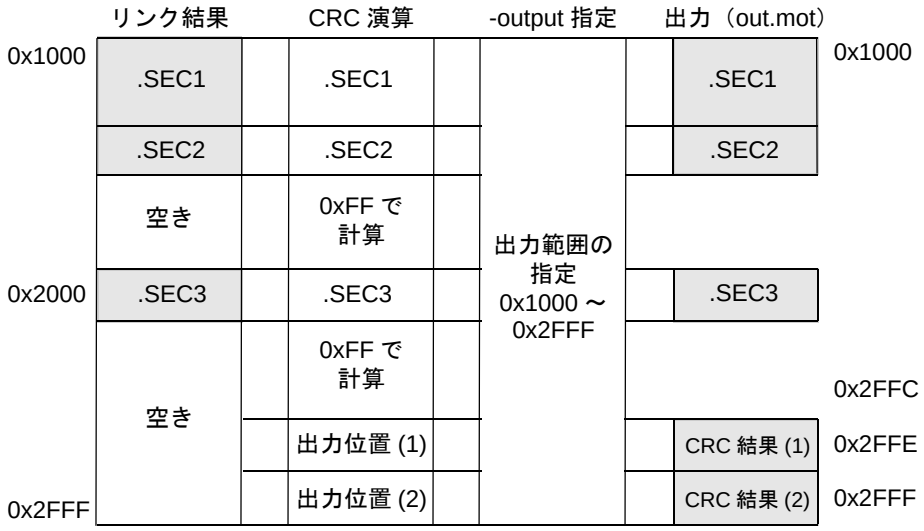


-crc オプション：-crc=1FFE=1000-1FFD,2000-2FFF
0x1000 ～ 0x1FFD と 0x2000 ～ 0x2FFF の領域に対して CRC 演算を行い，その結果を 0x1FFE 番地に出力します。
計算範囲にある空き領域は -space オプションを指定していない場合，-space=FF オプションを指定していると仮定して，CRC 演算を行います。

-output オプション：-output=flmem.mot=1000-1FFF
-space オプションを指定していないため，空き領域は flmem.mot に出力されません。
CRC 演算は，空き領域では 0xFF で計算を行いますが，0xFF を埋めることはありません。

例 4.

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



-crc オプション (1)：-crc=2FFC=1000-1FFF
0x1000 ～ 0x1FFF の領域に対して CRC 演算を行い，その結果を 0x2FFC 番地に出力します。
-crc オプション (2)：-crc=2FFE=2000-2FFB
0x2000 ～ 0x2FFB の領域に対して CRC 演算を行い，その結果を 0x2FFE 番地に出力します。

［備考］

- 本オプションは，複数のロード・モジュール・ファイルを入力した場合は無効となります。
- 【V1.06 以前】
本オプションは，-form={stype|hexadecimal} オプションを指定した場合のみ有効です。
【V1.07 以降】
本オプションは，-form={stype | hexadecimal | bin} オプションを指定した場合のみ有効です。
- -space オプションを指定していない場合，かつ計算範囲に出力されない空き領域がある場合，空き領域には 0xFF が設定されているものとして CRC の演算を行います。
- CRC 演算の計算範囲にオーバーレイ指定されている領域が含まれる場合，エラーとなります。
- RL78,78K0R C コンパイラ・パッケージ CA78K0R（別売）のオブジェクト・コンバータ（OC78K0R）と本製品の最適化リンカの -crc オプション指定との対応は，以下の様になります。

OC78K0R	最適化リンカ
HIGH	16-CCITT-MSB-LITTLE-4
HIGH(SENT)	SENT-MSB
GENERAL	16-CCITT-LSB

-VECT

ベクタテーブル・アドレスの空き領域にアドレスを設定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- アドレス未設定のベクタテーブル・アドレスに対して，オプション指定のアドレスを設定します。
- 本オプションを指定した場合，ソース上に割り込みハンドラの記述がなくても，ベクタテーブル・セクションを最適化リンカが作成し，ベクタテーブル・アドレスにアドレスを設定します。
- *symbol* には，対象関数の先頭に “_（アンダースコア）” を付けた外部名を指定してください。
- *address* には，設定するアドレスを 16 進数表記で指定してください。
- ベクタテーブルの 0x2，0x3 番地の領域への値の設定は，以下の優先度順に従って行われます。
  -rrm オプション > -debug_monitor オプション > アセンブリ・ソース・ファイルの指定
  > -vectn オプション > -vect オプション

[使用例]

- ベクタテーブル・アドレスの空き領域に _dummy のアドレスを設定します。

```
>rlink a.obj b.obj -vect=_dummy
```

[備考]

- ユーザがベクタテーブル・アドレス・セクションをソース・プログラムで作成している場合，ベクタテーブル・アドレスの自動生成は行わないため，本オプションは無効となります。
- {*symbol*|*address*} の記述で先頭を 0 と記述したものは，すべてアドレスとして判断します。
- 本オプションは，-form={object|relocate|library} オプション，-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。

-VECTN

特定ベクタテーブル・アドレスの領域にアドレスを設定します。

[指定形式]

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

- 省略時解釈
  なし

[詳細説明]

- ベクタテーブル・アドレスの特定アドレスに対して，オプション指定のアドレスを設定します。
- 本オプションを指定した場合，ソース上に割り込みハンドラの記述がなくても，ベクタテーブル・アドレス・セクションを最適化リンカが作成し，ベクタテーブル・アドレスにアドレスを設定します。
- vector-number には，16 進数で 0x0 ～ 0x7E の範囲で偶数を指定してください。
- *symbol* には，対象関数の先頭に “_（アンダースコア）” を付けた外部名を指定してください。
- *address* には，設定するアドレスを 16 進数表記で指定してください。
- ベクタテーブルの 0x2，0x3 番地の領域への値の設定は，以下の優先度順に従って行われます。
  -rrm オプション > -debug_monitor オプション > アセンブリ・ソース・ファイルの指定
  > -vectn オプション > -vect オプション
- -SPLIT_VECT オプションの指定が無いとき，-VECTN オプションで指定されない空き領域には，以下の優先度で値を設定します。
  1. VECT オプションで指定した値
  2. “__dummy_int” という名称（内部名）の定義シンボルがリンク対象内に存在すれば，そのシンボルのアドレス
  3.“dummy_int” という名称（内部名）の定義シンボルがリンク対象内に存在すれば，そのシンボルのアドレス
  4. 上記のいずれでもない場合は 0
- -SPLIT_VECT オプションの指定があるときは，-VECTN オプションで指定されない空き領域に対するベクタ番号別セクションは生成しません。

[使用例]

- ベクタテーブル・アドレス 0x14 に _dummy のアドレスを設定します。

```
>rlink a.obj b.obj -vectn=14=_dummy
```

[備考]

- ユーザがベクタテーブル・アドレス・セクションをソース・プログラムで作成している場合，ベクタテーブル・アドレスの自動生成は行わないため，本オプションは無効となります。
- 本オプションは，-form={object|relocate|library} オプション，-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。

-SPLIT_VECT 【V1.07 以降】

ベクタテーブル・セクションをベクタテーブル・アドレス別に分割して生成します。

[指定形式]

```
-SPLIT_VECT
```

- 省略時解釈
  なし

[詳細説明]

- ベクタテーブル・セクションをベクタテーブル・アドレス別に分割して生成します。
- ベクタテーブル・アドレスの空き領域に対しては，ベクタテーブル・セクションを生成しません。

[使用例]

- ベクタテーブル・アドレス 0x14 に対してベクタテーブル・セクション ".vect14" を生成します。

```
>rlink a.obj b.obj -vectn=14=__dummy -split_vect
```

[備考]

- 本オプションは，-vect オプションがあると無効になります。
- 本オプションは，-form={object | relocate | library} オプション，-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。

-VFINFO

変数 / 関数情報ファイルを出力します。

[指定形式]

```
-VFINFO[=file]                          【V1.04 以前】
-VFINFO[=[file][(attribute,...)]]  【V1.05 以降】
```

- 省略時解釈
  変数 / 関数情報ファイルを出力しません。

[詳細説明]

- *file* には，変数 / 関数情報ファイル名を指定してください。
- ファイル名を省略した場合は，変数 / 関数情報ファイルとして vfinfo.h を出力します。
- ファイル名に拡張子がない場合，".h" を拡張子とします。
- 出力属性 *attribute* に指定可能なものは，callt，near，rom_forbid，または far_forbid です。出力属性により，以下のように変数 / 関数情報ファイルを作成します。【V1.05 以降】

変数情報	
出力属性	内容
指定なし	saddr 領域の余剰分，参照回数の多い変数に対して #pragma saddr を出力します。 既に saddr 変数として宣言しているものは参照回数に関わらず saddr 変数を維持し，#pragma saddr の出力対象とせずコメントアウトして変数情報を出力します。
関数情報	
出力属性	内容
指定なし または callt	callt エントリまたは near 領域の余剰分，呼び出し回数の多い関数に対して #pragma callt を出力します。 既に callt 関数として宣言しているものは，呼び出し回数に関わらず callt 関数を維持し，#pragma callt の出力対象とせずコメントアウトして関数情報を出力します。
near	near 領域の余剰分，呼び出し回数の多い関数に対して #pragma near を出力します。 既に callt 関数または near 関数として宣言しているものは，呼び出し回数に関わらず #pragma near の出力対象とせずコメントアウトして関数情報を出力します。
callt,near	callt エントリまたは near 領域の余剰分，呼び出し回数の多い関数に対して #pragma callt を出力します。 既に callt 関数として宣言しているものは，呼び出し回数に関わらず callt 関数を維持し，#pragma callt の出力対象とせずコメントアウトして関数情報を出力します。 次に，残りの関数の中から，near 領域の余剰分，呼び出し回数の多い関数に対して #pragma near を出力します。既に near 関数として宣言しているものは，呼び出し回数に関わらず near 関数を維持し，#pragma near の出力対象とせずコメントアウトして関数情報を出力します。
rom_forbid	ROM オプションで指定したセクションの関数を，#pragma callt または #pragma near の対象から外します。
far_forbid	絶対アドレスで指定したセクション，-start オプションで far 領域に指定したセクションの関数を，#pragma callt または #pragma near の対象から外します。

［使用例］

- 変数 / 関数情報ファイルを file.h に出力します。

```
>rlink a.obj b.obj -vfinfo=file.h
```

［備考］

- 本オプションは，-form={object|relocate|library} オプション，-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。
- -device オプションを指定していない場合，本オプションは無効となります。
- セクションの配置アドレスが使用可能なアドレス範囲を超える場合，変数 / 関数情報ファイルには，配置可能な領域内に配置できたシンボル情報とセクション情報のみを出力します。【V1.04】
- セクション割り当てがオブジェクトファイル仕様領域外の場合はワーニングを出力し，外部変数割り付け情報ファイルを出力します。【V1.05 以降】

-CFI【Professional 版のみ】【V1.06 以降】

不正な間接関数呼び出し検出で用いる関数リストを生成します。

[指定形式]

```
-CFI
```

- 省略時解釈
  不正な間接関数呼び出し検出で用いる関数リストを生成しません。

[詳細説明]

- 不正な間接関数呼び出し検出で用いる関数リストを生成します。
  不正な間接関数呼び出し検出の詳細については，コンパイル・オプション「-control_flow_integrity【Professional 版のみ】【V1.06 以降】」を参照してください。
  リンカは，関数リストを .constf セクションに生成します。そのため，リンク時に -start オプションで .constf セクションを指定する必要があります。
- コンパイル時に -control_flow_integrity を指定して作られたオブジェクト・ファイルの場合，リンカは，コンパイラが自動抽出した情報を元に関数リストを生成します。
- コンパイル時に -control_flow_integrity を指定せずに作られたオブジェクト・ファイルの場合，リンカは，オブジェクト・ファイル内のリロケーション解決したシンボル全ての関数リストを生成します。
- 特定の関数を関数リストに追加する場合は，リンク・オプション -CFI_ADD_Func を指定してください。
  特定のオブジェクト・ファイル内の関数を関数リストから除外する場合は，リンク・オプション -CFI_IGNORE_Module を指定してください。

-CFI_ADD_Func【Professional 版のみ】【V1.06 以降】

不正な間接関数呼び出し検出で用いる関数リストに追加する関数シンボルまたはアドレスを指定します。

[指定形式]

```
-CFI_ADD_Func={symbol|address}[, ...]
```

- 省略時解釈
  なし

[詳細説明]

- 不正な間接関数呼び出し検出で用いる関数リストに，関数のシンボルまたはアドレスを登録します。
  不正な間接関数呼び出し検出の詳細については，コンパイル・オプション「-control_flow_integrity【Professional 版のみ】【V1.06 以降】」を参照してください。
- *address* は 16 進数で指定します。
- 指定した関数シンボルが（リンカで最適化を行った後の）ロード・モジュール内に存在しない場合，エラーとなります。
- 本オプションを複数回指定した場合は，指定した全ての関数シンボルまたはアドレスを関数リストに登録します。
- 本オプションを使用する場合，-CFI オプションの指定が必要です。-CFI オプションの指定がない場合はエラーとなります。

[使用例]

- C ソースの main 関数，関数アドレス 0x100，C ソースの sub 関数を関数リストに登録します。

```
>rlink -cfi -cfi_add_func=_main,100 -cfi_add_func=_sub a.obj b.obj
```

-CFI_IGNORE_Module【Professional 版のみ】【V1.06 以降】

不正な間接関数呼び出し検出で用いる関数リストから除外するオブジェクト・ファイルを指定します。

[指定形式]

```
-CFI_IGNORE_Module=suboption[, ...]
【V1.06 以前】
  suboption := file
【V1.07 以降】
  suboption := file[(module[, ...])]
```

- 省略時解釈
  なし

[詳細説明]

- 【V1.06 以前】
  不正な間接関数呼び出し検出で用いる関数リストから除外するオブジェクト・ファイルを指定します。
  【V1.07 以降】
  不正な間接関数呼び出し検出で用いる関数リストから除外するオブジェクト・ファイルとライブラリ・ファイルを指定します。ライブラリ・ファイル指定ではライブラリ内モジュール名を指定できます。
  不正な間接関数呼び出し検出の詳細については，コンパイル・オプション「-control_flow_integrity【Professional 版のみ】【V1.06 以降】」を参照してください。
- 指定したオブジェクト・ファイルが存在しない場合，エラーとなります。
- 本オプションを複数回指定した場合は，指定した全てのオブジェクト・ファイル内の関数を関数リストに登録しません。
- 本オプションを使用する場合，-CFI オプションの指定が必要です。-CFI オプションの指定がない場合はエラーとなります。

[使用例]

- a.obj，b.obj，c.obj 内の関数を関数リストから除外します。

```
>rlink -cfi -cfi_ignore_module=a.obj,b.obj -cfi_ignore_module=c.obj
```

- 【V1.07 以降】
  b.lib ライブラリ内の c モジュール内の関数を関数リストから除外します。

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

-RAM_INIT_TABLE_SECTION【V1.12 以降】

RAM 初期化のための情報テーブルを生成します。

[指定形式]

```
-RAM_INIT_TABLE_SECTION[= セクション名 ]
```

- 省略時解釈
  RAM 初期化のための情報テーブルを生成しません。

[詳細説明]

- RAM 初期化のための情報テーブルを生成します。
- 指定したセクション名に情報テーブルを生成します。
- セクション名の指定が無い場合は，セクション名 ".ram_init_table" で生成します。
- 一般にプログラムのスタートアップで .data セクションの内容を ROM から RAM に転送します。また，.bss セクションの配置 RAM を 0 でクリアします。このためのアドレスとサイズの情報をテーブルにして生成します。
- 情報テーブルは以下のフィールドを持つレコードが連続する形式になります。

	サイズ	内容
フィールド 1	4byte	ROM 先頭アドレス：ROM->RAM データ転送時 RAM 先頭アドレス：RAM クリア時 0：レコード終端時
フィールド 2	2byte	転送サイズ：ROM->RAM 転送または RAM クリア時 0：レコード終端時
フィールド 3	2byte	RAM 先頭アドレス：ROM->RAM 転送または RAM クリア時 0：レコード終端時

- テーブルの末端は，フィールド 1,2,3 すべてが 0 になります。
- スタートアップ処理で，テーブルの末端まで以下の処理をしてください。
  - フィールド 1 とフィールド 3 が異なるときは，フィールド 2 のサイズ分を，フィールド 1 からフィールド 3 へ，メモリコピーしてください。
  - フィールド 1 とフィールド 3 が同じときは，フィールド 2 のサイズ分を，フィールド 1 またはフィールド 3 から，メモリクリアしてください。
- 初期化対象のセクションは，CPU プログラム動作用のセクションで，FLEXIBLE APPLICATION ACCELERATOR (FAA) 用セクションなどはテーブルに含まれません。

[使用例]

- RAM 初期化のための情報テーブルを生成します。

```
> rlink a.obj b.obj -start=.data/1000,.dataR/f1000,.bss/f2000 -rom=.data=.dataR
-ram_init_table_section=.ram_init_table
```

- .data セクションサイズが 0x100 バイト，.bss セクションサイズが 0x200 バイトのとき，.ram_init_table セクションは以下のようなテーブルを生成します。

フィールド 1	フィールド 2	フィールド 3
0x01000	0x100	0x1000
0xf2000	0x200	0x2000
0	0	0

［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。

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。

- -VFINFO オプションと同時指定した場合は，セクションの配置アドレスが使用可能なアドレス範囲を超えてもリンク・マップ情報とシンボル情報を出力します。
  ただし，使用可能なアドレス範囲を超える場合は “**OVER**” と表示します。【V1.04 以降】

[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
VECTOR	指定不可	ベクタ情報を出力
STRUCT	指定不可	コンパイル時に -g オプション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シンボル情報内に出力します。（-form=relocate/object，-rename，-lib_rename，-hide，-compress，-optimize=symbol_delete オプションを指定した場合は無効）
RELOCATION_ATTRIBUTE	指定不可	再配置属性を出力
CFI	指定不可	-form=absolute オプション指定時 不正な間接関数呼び出し検出で用いる関数リストを出力 -form=hex/bin/stype オプション指定時かつ入力ファイルが absolute/hex/stype 以外 不正な間接関数呼び出し検出で用いる関数リストを出力
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,vector,struct 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,vector,struct 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	VECTOR	STRUCT	RELOCATION_ATTRIBUTE	CFI
-form=absolute	-show のみ	有効	有効	無効	無効	無効	無効	無効	無効	無効
	-show=all	有効	有効	無効	有効	有効	有効	有効	無効	無効
-form=library	-show のみ	有効	無効	有効	無効	無効	無効	無効	無効	無効
	-show=all	有効	無効	有効	無効	無効	無効	無効	無効	無効
-form=relocate	-show のみ	有効	無効	無効	無効	無効	無効	無効	無効	無効
	-show=all	有効	無効	無効	有効	無効	有効	無効	無効	無効
-form=object	-show のみ	無効	無効	無効	無効	無効	無効	無効	無効	無効
	-show=all	無効	無効	無効	無効	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	無効	無効	無効	無効
	-show=all	有効	有効	無効	有効	有効注 1	有効注 1	有効注 1	無効	無効

注 1. 入力ファイルがロード・モジュール・ファイル，インテル拡張ヘキサ・ファイル，モトローラ・S タイプ・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。

最適化

最適化オプションには，次のものがあります。

- -Optimize
- -NOOptimize
- -SEction_forbid
- -Absolute_forbid
- -SYmbol_forbid 【V1.02 以降】
- -ALLOW_OPTIMIZE_ENTRY_BLOCK【V1.13 以降】

-Optimize

モジュール間最適化の実行有無を指定します。

[指定形式]

```
-OPtimize[=Branch] 【V1.01】
-Optimize[={SYmbol_delete|Branch|SPeed|SAFe}] 【V1.02 以降】
```

- 省略時解釈
  すべての最適化を実行します。-optimize オプションの指定と同じです。

[詳細説明]

- コンパイル，アセンブル時に -goptimize オプションを指定したファイルに対して最適化を行います。
- サブオプション指定で，各最適化の実行有無を指定します。

パラメータ	意味	最適化対象プログラム	
		C 言語	アセンブリ言語
なし	すべての最適化を実行します。	○	×
symbol_delete 【V1.02 以降】	一度も参照のない変数 / 関数を削除します。コンパイル時に必ず entry オプションを指定してください。	○	×
branch	プログラムの配置に基づいて，分岐命令サイズを最適化します。 他の最適化項目を実行すると，指定の有無に関わらず必ず実行します。	○	○
speed 【V1.02 以降】	オブジェクトスピード低下を招く可能性のある最適化以外を実行します。 -optimize=symbol_delete, branch と同じです。	○	×
safe 【V1.02 以降】	変数や関数の属性によって制限される可能性のある最適化以外を実行します。 -optimize=branch と同じです。	○	×

[使用例]

- 分岐サイズの最適化を実施します。

```
>rlink a.obj b.obj -optimize=branch
```

[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。
- 実行アドレス（entry）を指定しない場合，-optimize=symbol_delete は無効になります。
- 無効なパラメータを指定した場合，警告を出力し無視します。

-NOOptimize

モジュール間最適化を行いません。

[指定形式]

```
-NOOPtimize
```

- 省略時解釈
  -optimize オプションの指定と同じです。

[詳細説明]

- モジュール間最適化を行いません。

[使用例]

- モジュール間最適化を行いません。

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

-SEction_forbid

特定セクションの最適化を抑止します。

[指定形式]

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

- 省略時解釈
  特定セクションの最適化を抑止しません。

[詳細説明]

- 特定セクションの最適化を抑止します。
- *sub* には，ファイル名，モジュール名，セクション名を指定します。
- 入力ファイル名，またはライブラリ・モジュール名を同時に指定することで，最適化抑止対象をセクション全体だけでなく，特定ファイルに限定することも可能です。

[使用例]

- .SEC1 セクションへの全最適化を抑止します。

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

- a.obj 内の .SEC1，.SEC2 セクションへの全最適化を抑止します。

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

[備考]

- 本オプションは，最適化を使用しないリンク処理では無効です。
- パスを記述した入力ファイルを最適化抑止する場合は，ファイル名にパスを記述してください。
- ミラー先領域の最適化を抑止する場合は，ミラー元領域を指定してください。

-Absolute_forbid

アドレス＋サイズの範囲の最適化を抑止します。

[指定形式]

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

- 省略時解釈
  アドレス＋サイズの範囲の最適化を抑止しません。

[詳細説明]

- アドレス＋サイズの範囲の最適化を抑止します。
- *address*，*size* には，アドレス，サイズを 16 進数で指定します。

[注意]

- 最適化実施前段階のセクションメモリ配置で，同一アドレスに重複して配置されていたとき，そのアドレスを最適化抑止 (absolute_forbid) に指定すると，重複しているすべてのセクションで最適化が実施されなくなります。これにより，期待する以上に最適化が行われなくなり，セクション配置のオーバーラップやオーバーフローとなることがあります。このような場合は，アドレス指定による最適化抑止 (absolute_forbid) をするのではなく，セクション指定による最適化抑止 (section_forbid) をしてください。

[使用例]

- アドレス 0x1000 ～ 0x11ff への最適化を抑止します。

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

[備考]

- 本オプションは，最適化を使用しないリンク処理では無効です。

-SYmbol_forbid 【V1.02 以降】

未参照シンボル削除最適化を抑止します。

[指定形式]

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

- 省略時解釈
  未参照シンボル削除最適化を抑止しません。

[詳細説明]

- 未参照シンボル削除最適化を抑止します。

[使用例]

- _func シンボルへの最適化を抑止します。

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

[備考]

本オプションは，最適化を使用しないリンク処理では無効です。

-ALLOW_OPTIMIZE_ENTRY_BLOCK【V1.13 以降】

実行開始シンボルより前に配置されている領域を最適化の対象にします。

[指定形式]

```
-ALLOW_OPTIMIZE_ENTRY_BLOCK
```

- 省略時解釈
  実行開始シンボルより前に配置されている領域を最適化の対象外にします。

[詳細説明]

- 実行開始シンボルより前に配置されている領域を最適化の対象にします。
- 本オプションを複数回指定した場合，１回指定した場合と同じ意味になります。このとき，警告を出力します。

[使用例]

- 実行開始シンボルより前に配置されている領域を含めて最適化を実施します。

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

[備考]

- 本オプションは，最適化を使用しないリンク処理では無効です。
- 本オプションは，-entry オプションでアドレスを指定した場合は警告を出力して無視します。
- 本オプションは，-entry オプションを指定していない場合は無効です。

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
- -USER_OPT_BYTE
- -OCDBG
- -SECURITY_OPT_BYTE【V1.12 以降】
- -SECURITY_ID
- -FLASH_SECURITY_ID【V1.12 以降】
- -AUTO_SECTION_LAYOUT
- -SPLIT_SECTION【V1.12 以降】
- -STRIDE_DSP_MEMORY_AREA【V1.12 以降】
- -DEBUG_MONITOR
- -RRM
- -SELF
- -SELFW
- -OCDTR
- -OCDTRW
- -OCDHPI
- -OCDHPIW
- -DSP_MEMORY_AREA【V1.12 以降】

-STARt

セクションの開始アドレスを指定します。

[指定形式]

```
-STARt=suboption[, ...]
  suboption := section-group[/address]
  section-group := section-list[: ...]
  section-list := section[, ...]
```

- 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。

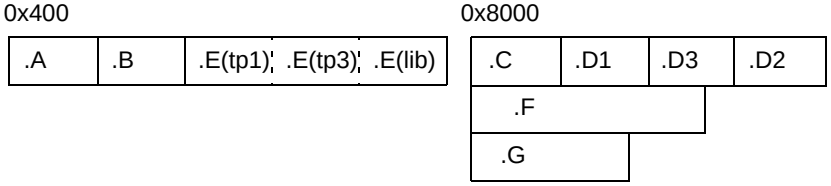
[詳細説明]

- 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)

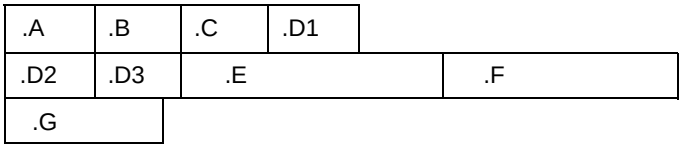
  - -start=.A,.B,.E/400,.C,.D*:.F:.G/8000 オプションを指定した場合



    - “:” 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

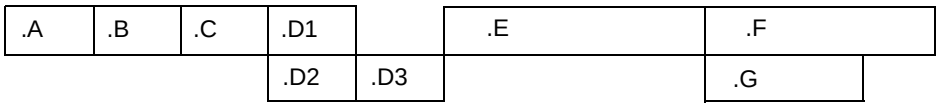
0x400



  - ":" 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。
- "()" を使用して本オプションを記述した場合，リンカの最適化機能は無効となります。

-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。

[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 .A，.B 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=.A,.B -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 = .A, .B

        ;SECTION NAME = .A
            .public _f
        _f .equ 0x00000000
            .public _g
        _g .equ 0x00000010
        ;SECTION NAME = .B
            .public _main
        _main .equ 0x00000020
```

[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。

-USER_OPT_BYTE

ユーザ・オプション・バイトに設定する値を指定します。

[指定形式]

```
-USER_OPT_BYTE=user-option-byte
```

- 省略時解釈
  本オプションを使用しない場合は，必ずアセンブリ・ソース・ファイルを使用してユーザ・オプション・バイト値を設定してください。

[詳細説明]

- ユーザ・オプション・バイトに設定する値 *user-option-byte* を指定します。
- ユーザ・オプション・バイト値は，デバイスにより値が異なります。設定する値は，デバイスのユーザーズ・マニュアルを参照してください。
  なお，省略時は，ユーザ・オプション・バイト値には，デバイス・ファイルの初期値が入ります。
- ユーザ・オプション・バイトは，16 進数で 0x0 ～ 0xFFFFFF の範囲で指定してください。
  3 バイトに満たない場合は，上位ビットに 0 を補てんします。
- ユーザ・オプション・バイトは，0xC0 ～ 0xC2 番地に設定されます。
  ユーザ・オプション・バイトの MSB 側からバイト単位で，0xC0 → 0xC2 の順で設定していきます。
- 0xC0 ～ 0xC2 番地のユーザ・オプション・バイト値は，アセンブリ・ソース・ファイル中に以下の再配置属性のセグメントを定義することでも指定可能です。
  ただし，0xC3 番地の制御値とあわせて 4 バイトで定義してください。

```
        .section    .option_byte, opt_byte
          .db       0xFD                ;0xC0 番地
          .db       0xFE                ;0xC1 番地
          .db       0xFF                ;0xC2 番地
          .db       0x04                ;0xC3 番地
```

- デバイス・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- アセンブリ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- デバイス・ファイルの指定があり，アセンブリ・ソース・ファイルの指定が重なった場合は，アセンブリ・ソース・ファイルの指定が優先されます。
- デバイス・ファイルの指定があり，アセンブリ・ソース・ファイルの指定と本オプションの指定がない場合は，デバイス・ファイルの初期値を利用するワーニングが出力されます。

[使用例]

- ユーザ・オプション・バイト値として 0xC0 番地に 0xFD，0xC1 番地に 0xFE，0xC2 番地に 0xFF を指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -user_opt_byte=FDFEFF
```

[備考]

- 本オプションは，-form={object|relocate|library} 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。
- -device オプションを指定していない場合，本オプションは無効となります。
- アセンブリ・ソース・ファイルにユーザ・オプション・バイト値を記述する場合，ラベル参照を記述しないでください。リロケーション解決が行われるため，期待する値にならない場合があります。

-OCDBG

オンチップ・デバッグ制御値に設定する値を指定します。

[指定形式]

```
-OCDBG=value
```

- 省略時解釈
  本オプションを使用しない場合は，必ずアセンブリ・ソース・ファイルを使用してオンチップ・デバッグ制御値を設定してください。

[詳細説明]

- オンチップ・デバッグ制御値に設定する値 *value* を指定します。
- オンチップ・デバッグ制御値は，デバイスにより値が異なります。設定する値は，デバイスのユーザーズ・マニュアルを参照してください。
  なお，省略時は，オンチップ・デバッグ制御値には，デバイス・ファイルの初期値が入ります。
- オンチップ・デバッグ制御値は，16 進数で 0x0 ～ 0xFF の範囲で指定してください。
- オンチップ・デバッグ制御値として指定できない値を指定した場合は，エラーとなります。
- オンチップ・デバッグ制御値は，0xC3 番地に設定されます。
- オンチップ・デバッグ制御値は，アセンブリ・ソース・ファイル中に以下の再配置属性のセグメントを定義することでも指定可能です。ただし，0xC0 番地からのユーザ・オプション・バイトとあわせて 4 バイトで定義してください。
  なお，デバイス・ファイルの指定と重なった場合は，アセンブリ・ソース・ファイルの指定が優先されます。

```
        .section    .option_byte, opt_byte
          .db       0xfd                ;0xC0 番地
          .db       0xfd                ;0xC1 番地
          .db       0xff                ;0xC2 番地
          .db       0x04                ;0xC3 番地
```

- デバイス・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- アセンブリ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。

[使用例]

- オンチップ・デバッグ制御値として 0xC3 番地に 0x04 を指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdbg=04
```

[備考]

- 本オプションは，-form={object|relocate|library} 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。
- -device オプションを指定していない場合，本オプションは無効となります。
- アセンブリ・ソース・ファイルにオンチップ・デバッグ制御値を記述する場合，ラベル参照を記述しないでください。リロケーション解決が行われるため，期待する値にならない場合があります。

-SECURITY_OPT_BYTE【V1.12 以降】

セキュリティ・オプション・バイト制御値に設定する値を指定します。

[指定形式]

```
-SECURITY_OPT_BYTE=value
```

- 省略時解釈
  アセンブリ・ソース・ファイルの指定した値か，デバイス・ファイルに定義された値を設定します。

[詳細説明]

- セキュリティ・オプション・バイト制御値 *value* を指定します。
- セキュリティ・オプション・バイト制御値は，16 進数で 0x0 ～ 0xFF の範囲で指定してください。
- セキュリティ・オプション・バイト制御値として指定できない値を指定した場合は，エラーとなります。
- セキュリティ・オプション・バイト制御値は，0xC4 番地に設定されます。
- セキュリティ・オプション・バイト制御値は，アセンブリ・ソース・ファイル中に以下の再配置属性のセグメントを定義することでも指定可能です。
  0xC0 番地からのユーザ・オプション・バイト，オンチップ・デバッグ制御値とあわせて 5 バイトで定義してください。

```
        .section    .option_byte, opt_byte
          .db       0xfd                ;0xC0 番地
          .db       0xfe                ;0xC1 番地
          .db       0xff                ;0xC2 番地
          .db       0x04                ;0xC3 番地
          .db       0xff                ;0xC4 番地
```

- アセンブリ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- デバイス・ファイルの指定とアセンブラ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションを優先します。

[使用例]

- セキュリティ・オプション・バイト制御値として 0xC4 番地に 0x04 を指定します。

```
>rlink a.obj b.obj -device=dr7f124fgj.dvf -security_opt_byte=04
```

[備考]

- 本オプションは，-form={object|relocate|library} 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。
- -device オプションを指定していない場合，本オプションは無効となります。
- アセンブリ・ソース・ファイルにセキュリティ ID 値を記述する場合，ラベル参照を記述しないでください。リロケーション解決が行われるため，期待する値にならない場合があります。

-SECURITY_ID

セキュリティ ID 値を指定します。

[指定形式]

```
-SECURITY_ID=value
```

- 省略時解釈
  セキュリティ ID 値を設定しません。

[詳細説明]

- セキュリティ ID 値 *value* を指定します。
- セキュリティ ID 値は，16 進数で指定してください。
- セキュリティ ID 値として指定できない値を指定した場合は，エラーとなります。
  ユーザ・オプション・バイトの MSB 側からバイト単位で，下位アドレスから上位アドレスの順で設定していきます。
- セキュリティ ID 値の配置先とセキュリティ ID の最大サイズは，デバイス・ファイルにより設定されます。
- (V1.11 以降) オプションの指定はデバイス仕様によるセキュリティ ID の最大サイズ以内で指定してください。
  (V1.10 以前) 10 バイト以内で指定してください。
  デバイス仕様によるセキュリティ ID の最大サイズを超えた場合はエラーになります。
  指定するセキュリティ ID のサイズが最大サイズに満たない場合は，上位ビットに 0 を補てんします。
- セキュリティ ID 値は，アセンブリ・ソース・ファイル中に以下の再配置属性のセグメントを定義することでも指定可能です。
  ただし，オプションの指定と同様にデバイス仕様によるセキュリティ ID サイズで定義してください。

```
        .section    .security_id, SECUR_ID
          .db       0x01            ;0xC4 番地
          .db       0x02            ;0xC5 番地
          .db       0x03            ;0xC6 番地
          .db       0x04            ;0xC7 番地
          .db       0x05            ;0xC8 番地
          .db       0x06            ;0xC9 番地
          .db       0x07            ;0xCA 番地
          .db       0x08            ;0xCB 番地
          .db       0x09            ;0xCC 番地
          .db       0x0A            ;0xCD 番地
```

- アセンブリ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- セキュリティ ID 値 は，必ずデバイスのユーザーズ・マニュアルを参照して設定を行ってください。

[使用例]

- セキュリティ ID 値を指定します。0xC4 番地から，0x01，0x02，0x03，0x04，0x05，0x06，0x07，0x08，0x09，0x0A を指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -security_id=0102030405060708090A
```

[備考]

- 本オプションは，-form={object|relocate|library} 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。
- -device オプションを指定していない場合，本オプションは無効となります。

- アセンブリ・ソース・ファイルにセキュリティ ID 値を記述する場合，ラベル参照を記述しないでください。リロケーション解決が行われるため，期待する値にならない場合があります。

-FLASH_SECURITY_ID【V1.12 以降】

フラッシュ・プログラマ・セキュリティ ID 値に設定する値を指定します。

[指定形式]

```
-FLASH_SECURITY_ID=value
```

- 省略時解釈
  フラッシュ・プログラマ・セキュリティ ID 値を設定しません。

[詳細説明]

- フラッシュ・プログラマ・セキュリティ ID 値 *value* を指定します。
- フラッシュ・プログラマ・セキュリティ ID 値は，16 進数で指定してください。
- フラッシュ・プログラマ・セキュリティ ID 値として指定できない値を指定した場合は，エラーとなります。
  フラッシュ・プログラマ・セキュリティ ID 値の MSB 側からバイト単位で，下位アドレスから上位アドレスの順で設定していきます。
- フラッシュ・プログラマ・セキュリティ ID 値の配置先とセキュリティ ID の最大サイズは，デバイス・ファイルにより設定されます。
- オプションの指定はデバイス仕様によるセキュリティ ID の最大サイズ以内で指定してください。
  デバイス仕様によるセキュリティ ID の最大サイズを超えた場合はエラーになります。
  指定するセキュリティ ID のサイズが最大サイズに満たない場合は，上位ビットに 0 を補てんします。
- セキュリティ ID 値は，アセンブリ・ソース・ファイル中に以下の再配置属性のセグメントを定義することでも指定可能です。
  ただし，オプションの指定と同様にデバイス仕様によるセキュリティ ID の最大サイズ以内で定義してください。

```
        .section  .flash_security_id, FLASH_SECUR_ID
          .db       0x01            ;0xD6 番地
          .db       0x02            ;0xD7 番地
          .db       0x03            ;0xD8 番地
          .db       0x04            ;0xD9 番地
          .db       0x05            ;0xDA 番地
          .db       0x06            ;0xDB 番地
          .db       0x07            ;0xDC 番地
          .db       0x08            ;0xDD 番地
          .db       0x09            ;0xDE 番地
          .db       0x0A            ;0xDF 番地
          .db       0x0B            ;0xE0 番地
          .db       0x0C            ;0xE1 番地
          .db       0x0D            ;0xE2 番地
          .db       0x0E            ;0xE3 番地
          .db       0x0F            ;0xE4 番地
          .db       0x10            ;0xE5 番地
```

- アセンブリ・ソース・ファイルの指定と本オプションの指定が重なった場合は，本オプションが優先されます。
- セキュリティ ID 値は，必ずデバイスのユーザーズ・マニュアルを参照して設定を行ってください。

[使用例]

- セキュリティ ID 値を指定します。0xD6 番地から，0x01，0x02，0x03，0x04，0x05，0x06，0x07，0x08，0x09，0x0A，0x0B，0x0C，0x0D，0x0E，0x0F，0x10 を指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -security_id=0102030405060708090A0B0C0D0E0F10
```

［備考］

- 本オプションは，-form={object|relocate|library} 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。
- -device オプションを指定していない場合，本オプションは無効となります。
- アセンブリ・ソース・ファイルにセキュリティ ID 値を記述する場合，ラベル参照を記述しないでください。リロケーション解決が行われるため，期待する値にならない場合があります。

-AUTO_SECTION_LAYOUT

セクションを自動的に配置します。

[指定形式]

```
-AUTO_SECTION_LAYOUT
```

- 省略時解釈
  セクションを自動的に配置しません。

[詳細説明]

- デバイス・ファイルの情報からセクションを自動的に配置します。
- -start オプションで指定されたセクション，および絶対アドレス・セクションを配置した後に，残りのセクションをデバイス・ファイルの情報から自動的に配置します。
- -cpu オプションが指定された場合，-device オプションで指定したデバイスのメモリ領域で，かつ -cpu オプションで指定したアドレス範囲に自動的に配置します。

[使用例]

- デバイス・ファイルの情報からセクションを自動的に配置します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -auto_section_layout
```

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

-SPLIT_SECTION【V1.12 以降】

セクションの自動配置で配置単位をモジュール別のセクション単位で行います。

［指定形式］

```
-SPLIT_SECTION
```

- 省略時解釈
  セクションの自動配置を，結合したセクション単位で行います。

［詳細説明］

- SPLIT_SECTION を指定したとき，セクションはモジュール別で分かれたまま，デバイスの配置可能な領域へ個々に割り当てます。
  指定しない場合のセクションの自動配置は，リンクするモジュールから同じセクションを結合して，一つのセクションとしたあとにデバイスの配置可能な領域に割り当てます。

  以下のように file1.obj と file2.obj に text_section と data_section があるとします。

  file1.obj

text_section サイズ 1K
data_section サイズ 1K

  file2.obj

text_section サイズ 1K
data_section サイズ 1K

デバイスが ROM サイズ 3K と RAM-A サイズ 1.5K RAM-B サイズ 1.5K で構成されていた場合，auto_section_layout によるセクションの自動割り当ては以下のようになります。
ROM（サイズ 3K）は十分なサイズがあるので file1.obj と file2.obj の両方の text_section を配置できます。
RAM は RAM-A（サイズ 1.5K）と RAM-B（サイズ 1.5K）に分割され，file1.obj と file2.obj の両方を結合した data_section（サイズ 2K）を下図のように配置できません。

メモリ	セクション配置
ROM サイズ 3K	セクション text_section サイズ 2K file1.obj: text_section サイズ 1K file2.obj: text_section サイズ 1K
	空きサイズ 1K

	メモリ未配置
RAM-A サイズ 1.5K	セクション data_section サイズ 2K file1.obj: data_section file2.obj: data_section
	割り当て失敗 0.5K 溢れる

	メモリ未配置
RAM-B サイズ 1.5K	空きサイズ 1.5K

SPLIT_SECTION を指定しているなら，RAM-A（サイズ 1.5K）に file1.obj の data_section（サイズ 1K）を配置したあと，RAM-B（サイズ 1.5K）に file2.obj の data_section（サイズ 1K）を下図のように配置できます。

メモリ	セクション配置
ROM サイズ 3K	セクション text_section サイズ 1K file1.obj: text_section サイズ 1K
	セクション $text_section_part01 サイズ 1K file2.obj: text_section サイズ 1K
	空きサイズ 1K

	メモリ未配置
RAM-A サイズ 1.5K	セクション data_section サイズ 1K file1.obj: data_section サイズ 1K
	空きサイズ 0.5K

	メモリ未配置
RAM-B サイズ 1.5K	セクション $data_section_part01 サイズ 1K file2.obj: data_section サイズ 1K
	空きサイズ 0.5K

モジュール別セクションは以下のセクション名になります。

```
$sss_part??
```

sss ：分割前セクション名
?? ：01 ～ 99

［使用例］

- モジュール別のセクションを配置します。

```
>rlink a.obj b.obj -device=dr5f100pj.dvf -auto_section_layout -split_section
```

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

-STRIDE_DSP_MEMORY_AREA【V1.12 以降】

自動配置のセクション割り付けは，FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域で分断した領域に分けて配置します。

［指定形式］

```
-STRIDE_DSP_MEMORY_AREA
```

- 省略時解釈
  セクションの自動配置を，FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域で分割しません。

［詳細説明］

- STRIDE_DSP_MEMORY_AREA を指定したとき，セクションを順番に，FLEXIBLE APPLICATION ACCELERATOR (FAA) メモリ領域の手前まで配置していき，残りは FLEXIBLE APPLICATION ACCELERATOR (FAA) メモリ領域の後ろから，別セクションとして配置します。

  以下のように file1.obj,file2.obj,file3.obj と file4.obj に data_section があるとします。

  file1.obj

data_section サイズ 1K

  file2.obj

data_section サイズ 1K

  file3.obj

data_section サイズ 1K

  file4.obj

data_section サイズ 1K

デバイスが RAM-A サイズ 2.5K RAM-B サイズ 2.5K で構成されていた場合，auto_section_layout によるセクションの自動割り当ては以下のようになります。
RAM は RAM-A（サイズ 2.5K）と RAM-B（サイズ 2.5K）に分割され，file1.obj,file2.obj,file3.obj と file4.obj を結合した data_section（サイズ 4K）を下図のように配置できません。

メモリ	セクション配置
RAM-A サイズ 2.5K	セクション data_section: サイズ 4K file1.obj: data_section サイズ 1K file2.obj: data_section サイズ 1K file3.obj: data_section サイズ 1K
DSP 用メモリ領域	file4.obj: data_section サイズ 1K
	割り当て失敗 1.5K 溢れる
RAM-B サイズ 2.5K	空きサイズ 2.5K

STRIDE_DSP_MEMORY_AREA を指定しているなら，RAM-A（サイズ 2.5K）に file1.obj と file2.obj の data_section（サイズ 2K）を配置したあと，RAM-B（サイズ 2.5K）に file3.obj と file3.obj の data_section（サイズ 2K）を下図のように配置できます。

メモリ	セクション配置
RAM-A サイズ 2.5K	セクション data_section: サイズ 2K file1.obj: data_section サイズ 1K file2.obj: data_section サイズ 1K
	空きサイズ 0.5K
DSP 用メモリ領域	
RAM-B サイズ 2.5K	セクション $data_section_part01: サイズ 2K file3.obj: data_section サイズ 1K file4.obj: data_section サイズ 1K
	空きサイズ 0.5K

このときの，分割して配置する別セクションは以下のセクション名になります。

```
$sss_part??
```

sss ：分割前のセクション名
?? ：01 ～ 99

［使用例］

- FLEXIBLE APPLICATION ACCELERATOR (FAA) メモリ領域で分割して配置します。

```
> rlink a.obj b.obj -device=dr5f10y14.dvf -auto_section_layout -dsp_memory_area
-stride_dsp_memory_area
```

［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。
- split_section 指定時，本オプションはエラーを出力して終了します。
- auto_section_layout 指定がないとき，ワーニングを出力して無視します。
- dsp_memory_area 指定がないとき，ワーニングを出力して無視します。

-DEBUG_MONITOR

OCD モニタのメモリ領域を指定します。

[指定形式]

```
-DEBUG_MONITOR[=address1-address2]
```

- 省略時解釈
  OCD モニタのメモリ領域を確保しません。

[詳細説明]

- *address1*，*address2* には，OCD モニタ先頭アドレス，OCD モニタ終了アドレスを指定します。
- OCD モニタ先頭アドレス，OCD モニタ終了アドレスを省略した場合，以下を指定したものとします。
  - 8 ビット CPU の場合
    0 バイト
  - 8 ビット CPU 以外の場合
    OCD モニタ先頭アドレス：内蔵 ROM の終了アドレス - 512 + 1
    OCD モニタ終了アドレス：内蔵 ROM の終了アドレス
- 【V1.10 以前】0x2，0x3，0xCE ～ 0xD7 番地と，OCD モニタ先頭アドレスから OCD モニタ終了アドレスの領域を 0xFF で充てんします。
- 【V1.11 以降】0x2，0x3 番地，とデバイス仕様にあわせた領域，および，OCD モニタ先頭アドレスから OCD モニタ終了アドレスの領域を 0xFF で充てんします。
- ベクタテーブルでもある 0x2，0x3 番地の領域は，以下の指定を優先します。
  -rrm オプション > -debug_monitor オプション > アセンブリ・ソース・ファイルの指定
  > -vectn オプション > -vect オプション
- OCD モニタのメモリ領域がユーザ・オプション・バイトの配置先を含む場合は，以下の指定を優先します。
  -user_opt_byte オプション > アセンブリ・ソース・ファイルの指定 > デバイス・ファイルの指定
  > -debug_monitor オプション
- OCD モニタのメモリ領域がオンチップ・デバッグ制御値の配置先を含む場合は，以下の指定を優先します。
  -ocdbg オプション > アセンブリ・ソース・ファイルの指定 > デバイス・ファイルの指定
  > -debug_monitor オプション
- OCD モニタのメモリ領域がセキュリティ ID 値の配置先を含む場合は，以下の指定を優先します。
  -security_id オプション > アセンブリ・ソース・ファイルの指定 > デバイス・ファイルの指定
  > -debug_monitor オプション

[使用例]

- OCD モニタのメモリ領域を 0x300 ～ 0x3FF に指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -debug_monitor=300-3FF
```

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

-RRM

RRM/DMM 機能用ワーク領域を指定します。

[指定形式]

```
-RRM=address
```

- 省略時解釈
  RRM/DMM 機能用ワーク領域を確保しません。

[詳細説明]

- *address* には，RRM/DDM 機能用ワーク領域の先頭アドレスを指定します。
- 先頭アドレスから 4 バイトを RRM/DDM 機能用ワーク領域とします。
- -debug_monitor オプション指定により確保された 0x2，0x3 番地に RRM 先頭アドレスを設定します。
- 【V1.13 以降】RRM 先頭アドレスから 4 バイトに，セクションを配置しません。
- ベクタテーブルでもある 0x2，0x3 番地の領域は，以下の指定を優先します。
  -rrm オプション > -debug_monitor オプション > アセンブリ・ソース・ファイルの指定
  > -vectn オプション > -vect オプション

[使用例]

- 0xFFDE0 番地から 4 バイトを RRM/DDM 機能用ワーク領域に指定します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -debug_monitor=2FC00-2FFF -rrm=FFDE0
```

[備考]

- 本オプションは，以下のいずれかに該当する場合に無効となります。
  - -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
  - -device オプションで指定したデバイスが，対応デバイスでない場合
  - -debug_monitor オプションを指定しなかった場合

-SELF

セルフ RAM 領域にセクションを配置しません。

[指定形式]

```
-SELF
```

- 省略時解釈
  セルフ RAM 領域はセクション配置の対象となります。

[詳細説明]

- セルフ RAM 領域にセクションを配置しません。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- セルフ RAM 領域にセクションを配置しません。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -self
```

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

-SELFW

セルフ RAM 領域にセクションを配置した場合は警告を出力します。

[指定形式]

```
-SELFW
```

- 省略時解釈
  セルフ RAM 領域に対するチェックは行いません。

[詳細説明]

- セルフ RAM 領域にセクションを配置した場合は警告を出力します。
- セルフ RAM 領域をまたいでセクションを配置した場合はエラーとなります。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- セルフ RAM 領域にセクションを配置した場合は警告を出力します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -selfw
```

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

-OCDTR

トレース RAM とセルフ RAM 領域にセクションを配置しません。

[指定形式]

```
-OCDTR
```

- 省略時解釈
  トレース RAM とセルフ RAM 領域はセクション配置の対象となります。

[詳細説明]

- トレース RAM とセルフ RAM 領域にセクションを配置しません。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- トレース RAM とセルフ RAM 領域にセクションを配置しません。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdtr
```

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

-OCDTRW

トレース RAM とセルフ RAM 領域にセクションを配置した場合は警告を出力します。

[指定形式]

```
-OCDTRW
```

- 省略時解釈
  トレース RAM とセルフ RAM 領域に対するチェックは行いません。

[詳細説明]

- トレース RAM とセルフ RAM 領域にセクションを配置した場合は警告を出力します。
- トレース RAM とセルフ RAM 領域をまたいでセクションを配置した場合はエラーとなります。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- トレース RAM とセルフ RAM 領域にセクションを配置した場合は警告を出力します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdtrw
```

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

-OCDHPI

ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置しません。

[指定形式]

```
-OCDHPI
```

- 省略時解釈
  ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域はセクション配置の対象となります。

[詳細説明]

- ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置しません。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置しません。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdhpi
```

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

-OCDHPIW

ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置した場合は警告を出力します。

[指定形式]

```
-OCDHPIW
```

- 省略時解釈
  ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域に対するチェックは行いません。

[詳細説明]

- ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置した場合は警告を出力します。
- ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域をまたいでセクションを配置した場合はエラーとなります。
- __STACK_ADDR_START シンボルと __STACK_ADDR_END シンボルは，saddr 領域を除いて設定します。

[使用例]

- ホット・プラグイン RAM，トレース RAM，およびセルフ RAM 領域にセクションを配置した場合は警告を出力します。

```
>rlink a.obj b.obj -device=dr5f10y14.dvf -ocdhpiw
```

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

-DSP_MEMORY_AREA【V1.12 以降】

FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域にセクションを配置しません。

[指定形式]

```
-dsp_memory_area
```

- 省略時解釈
  FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域はセクション配置の対象となります。

[詳細説明]

- FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域にセクションを配置しません。

[使用例]

- FLEXIBLE APPLICATION ACCELERATOR (FAA) と共用するメモリ領域にセクションを配置しません。

```
>rlink a.obj b.obj -device=dr5f100pj.dvf -auto_section_layout -dsp_memory_area
```

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

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
- -CHECK_DEVICE
- -CHECK_64K_ONLY
- -NO_CHECK_SECTION_LAYOUT
- -CHECK_OUTPUT_ROM_AREA 【V1.07 以降】

-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=suboption[, ...]
  suboption := type=address1-address2
```

- 省略時解釈
  セクションの割り付けアドレスの指示からの整合性をチェックしません。

[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]

【V1.01 以前】

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

【V1.02 以降】

- 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。

-CHECK_DEVICE

オブジェクト・ファイルの作成時に指定したデバイス・ファイルのチェックを行います。

[指定形式]

```
-CHECK_DEVICE
```

- 省略時解釈
  デバイス・ファイルのチェックを行いません。

[詳細説明]

- オブジェクト・ファイルの作成時に指定したデバイスファイル，および，-device オプションで指定したデバイス・ファイルがすべて同一であることをチェックします。
- 異なるデバイスファイルがある場合はエラーとなります。

[使用例]

- オブジェクト・ファイルの作成時に指定したデバイスファイル，および，-device オプションで指定したデバイス・ファイルがすべて同一であることをチェックします。

```
>rlink a.obj b.obj -check_device -device=dr5f10y14.dvf
```

[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。

-CHECK_64K_ONLY

セクションが（64K-1）バイト境界をまたいで配置されるかをチェックしません。

[指定形式]

```
-CHECK_64K_ONLY
```

- 省略時解釈
  セクションが 64K バイト境界，または（64K-1）バイト境界をまたいで配置された場合にエラーとなります。

[詳細説明]

- セクションが（64K-1）バイト境界をまたいで配置されるかをチェックしません。
- セクションが 64K バイト境界をまたいで配置された場合はエラーとなります。
- 以下の再配置属性を持つセクションが 64K バイト境界，および（64K-1）バイト境界のチェックの対象となります。

再配置属性	デフォルト・セクション名
TEXTF_UNIT64KP	.textf_unit64kp
CONST	.const
CONSTF	.constf
DATA	.data
BSS	.bss
DATAF	.dataf
BSSF	.bssf

- セクションが 64K バイト境界をまたいで配置されるとは，セクションのアドレスの下位 16 ビットが 0xFFFF を超えて 0x0000 に続くことを意味します。
- セクションが（64K-1）バイト境界をまたいで配置されるとは，セクションのアドレスの下位 16 ビットが 0xFFFE を超えて 0xFFFF に続くことを意味します。

[使用例]

- セクションが（64K-1）バイト境界をまたいで配置されるかをチェックしません。

```
>rlink a.obj b.obj -check_64k_only
```

[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。

-NO_CHECK_SECTION_LAYOUT

セクションのアドレス割り付けをチェックしません。

[指定形式]

```
-NO_CHECK_SECTION_LAYOUT
```

- 省略時解釈
  セクションが以下のようにメモリ配置されているかチェックし，配置されていない場合にエラーとなります。

セクション	配置先
.option_byte	アドレス固定
.security_id	アドレス固定
.sbss / .sdata	saddr 領域
.sbss / .sdata （-rom オプション指定時，RAM 側）	saddr 領域
.bss / .data	内部 RAM
.bss / .data（-rom オプション指定時，RAM 側）	内部 RAM
.const	Flash ミラー空間
.constf	内部 ROM
.sdata / .data（-rom オプション指定時，ROM 側）	内部 ROM
.text	プログラムメモリ
.text_unit64kp / .textf	プログラムメモリ

[詳細説明]

- デバイス・ファイルから読み込んだメモリ配置と，セクションのメモリ配置が整合するかのチェックを行いません。

[使用例]

- セクションのアドレス割り付けをチェックしません。

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

[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。

-CHECK_OUTPUT_ROM_AREA 【V1.07 以降】

ヘキサ・ファイルの出力アドレス範囲が，内部 ROM またはデータフラッシュにあることをチェックします。

[指定形式]

```
-CHECK_OUTPUT_ROM_AREA
```

- 省略時解釈
  チェックしません。

[詳細説明]

- 本オプションを指定した時，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・ファイルのアドレス範囲が内部 ROM またはデータフラッシュにあることをチェックします。内部 ROM またはデータフラッシュの範囲外にデータがあるときはワーニングを出力します。

[使用例]

- インテル拡張ヘキサ・ファイルの出力データのアドレスが内部 ROM またはデータフラッシュの範囲外にあるかチェックします。

```
>rlink -form=hex a.obj b.obj -device=dr5f100pj.dvf -check_output_rom_area
```

[備考]

- 本オプションは -device オプションがない場合は無効になります。
- 本オプションは -form={hexadecimal | stype} オプションがない場合は無効になります。

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 sub.txt 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 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，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
```

マイコン指定

マイコン指定オプションには，次のものがあります。

- -DEVICE

-DEVICE

リンク時に使用するデバイス・ファイル名を指定します。

[指定形式]

```
-DEVICE=file
```

- 省略時解釈
  リンク時にデバイス・ファイルの情報を使用しません。
  本オプション省略時には，ユーザ・オプション・バイト指定，オンチップ・デバッグ制御値指定，セキュリティ ID 値指定は無効となります。

[詳細説明]

- リンク時に使用するターゲット・デバイス・ファイル名 *file* を指定します。
- 指定したファイルが存在しない場合は，エラーとなります。
- 指定したターゲット・デバイス・ファイルの情報に対応したオブジェクト・コードを生成します。
- コンパイラに指定した CPU コアや乗除積和演算器の使用，およびアセンブラに指定した CPU コアと，デバイス・ファイルの内容が異なる場合，エラーとなります。
- アセンブラに指定したミラー領域範囲と，デバイス・ファイルが指定するミラー領域範囲が異なる場合，エラーとなります。

[使用例]

- ターゲット・デバイス・ファイル名 DR5F10Y14.DVF を指定します。

```
>rlink file1.obj -device=dr5f10y14.dvf
```

[備考]

- デバイス・ファイル指定が無い場合は，saddr 領域外に saddr 変数用セクションを配置してもエラーを出力しません。

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
- -LIB_REName 【V1.08 以降】
- -DELete
- -REPlace
- -EXTract
- -STRip
- -CHange_message
- -Hide
- -Total_size
- -VERBOSE 【V1.10 以降】
- -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・ポイント・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]

- S9 レコードを終端に出力します。

```
>rlink a.obj b.obj -form=stype -output=c.mot -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 -output=c.abs -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 -output=c.abs -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 -output=c.abs -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
  -memory=High 指定時と同じ処理を行います。

[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	リンク時に必要な情報のロードをまとめて行い，処理速度を優先します。
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 -nooptimize -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，-lib_rename，-stack，-optimize オプションのいずれか
    -list オプションと -show[={reference|xreference|struct|all}] オプション
  - -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，-allow_duplicate_module_name 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，「表 2.9 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=suboption[, ...]
  suboption := {(names)|file(names)|module(names)}
  names := 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。
- *file* を指定することで，*file* に含まれるセクションだけ名前を変更することができます。
- ライブラリ出力時（-form=library 指定時）には，*module* を指定することで，入力ライブラリ内の *module* に含まれるセクションだけ名前を変更することができます。
  それ以外の場合で入力ライブラリ内のセクション名を変更する場合は，-lib_rename オプションを使用してください。
- *file*，*module* を指定して，それらに含まれる外部シンボルだけ名前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。
- 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。
- 本オプションを複数回指定した場合，すべての指定が有効になります。
- 次の場合はエラーとなります。
  - 指定した *name*，*file*，*module* がみつからない場合。

[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 .SEC1 セクションを .SEC2 セクションに変更します。

```
>rlink -form=library -library=lib1.lib -rename=(.SEC1=.SEC2)
```

[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エラーとなります。
- -form={absolute|hexadecimal|stype|binary} 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 -Omerge_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。

-LIB_RENname 【V1.08 以降】

ライブラリから入力されたシンボルやセクションの名前を変更します。

[指定形式]

```
-LIB_RENname=suboption[, ...]
  suboption := (names)
             |  file(names)
             |  file|modules(names)
  modules := module[|module ...]
  names := name1=name2[, ...]
```

- 省略時解釈
  なし

[詳細説明]

- -library オプションで指定したライブラリ内モジュールに含まれる外部シンボル名，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。
- 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。
- 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。
- 本オプションを複数回指定した場合，すべての指定が有効になります。
- 次の場合はエラーとなります。
  - 指定した *name*，*file*，*module* がみつからない場合。
  - パラメータを省略した場合。

[使用例]

- b.lib にある _sym1 を _data に変更します。

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

[備考]

- 本オプションは，-form={object,library} オプション，-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エラーとなります。
- -form={absolute|hexadecimal|stype|binary} オプションを指定した場合は，-show=struct オプションを同時に指定できません。
- 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 -Omerge_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=suboption[, ...]
  suboption := {(symbol[, ...])|file(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 -Omerge_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=suboption[, ...]
  suboption := {file|file(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=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 -Omerge_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。
- 複数の入力ファイルを指定することはできません。

[使用例]

- file1.abs のデバッグ情報を削除し，file1.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 にバックアップします。

```
>rlink -strip file1.abs
```

[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=suboption[, ...]
  suboption := {level|level=range[, ...]}
  range := {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。
  なお，エントリ関数名は秘匿対象にはなりません。
  - 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

[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。
- コンパイル，アセンブル時に -goptimize オプションを指定したファイルを入力する際，出力ファイル形式がリロケータブル・ファイル，またはライブラリ・ファイルの場合は，本オプションを指定することはできません。
- 外部変数アクセス最適化を行う状況で本オプションを指定する場合は，一度目のリンク時には指定せず，二度目のリンク時にのみ指定してください。
- 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 オプションを使用する場合，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。
- 本オプションは，-form={object|relocate|library} 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。

-VERBOSE 【V1.10 以降】

詳細情報を標準エラー出力に表示します。

[指定形式]

```
-VERBOSE=<sub>[, ...]
sub : CRC
```

- 省略時解釈
  なし

[詳細説明]

- サブオプションに指定した内容を標準エラー出力に表示します。
- サブオプションに指定可能なものを以下に示します。

CRC	CRC の演算結果，および出力位置アドレスを表示します。 crc オプションを指定したときに有効です。

[使用例]

- CRC の演算結果，および出力アドレスを標準エラー出力に表示します。

```
> rlink a.obj -form=stype -start=.SEC1/1000 -crc=2000=1000-10ff/CCITT -verbose=crc
```

-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 sub.txt を以下の内容で作成します。

```
input=a.obj,b.obj                          ;(1)
start=.SEC1,.SEC2,.SEC3/100,.SEC4/8000  ;(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 sub.txt を指定します。

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

（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 sub.txt を以下の内容で作成します。

```
input=a.obj,b.obj                          ;(1)
start=.SEC1,.SEC2,.SEC3/100,.SEC4/8000  ;(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 sub.txt を指定します。

```
>rlink -subcommand=sub.txt -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。

2.5.4 ライブラリジェネレータ ・オプション【V1.13.00 以降】

ここではライブラリ生成・フェーズのオプションについて説明します。

表 2.10 ライブラリジェネレータ・オプション

分類	オプション	説明
ライブラリ生成制御	-head	構築対象のライブラリを指定します。
	-lang	標準ライブラリの言語規格を指定します。
	-secure_malloc【Professional 版のみ】	セキュリティ機能用 malloc 系ライブラリを生成します。
出力制御	-output	出力ファイル名を指定します。
その他	-logo / -nologo	コピーライト表示を制御します。
	-subcommand	サブコマンド・ファイルを指定します。

ライブラリ生成制御

ライブラリ生成制御オプションには，次のものがあります。

- -head
- -lang
- -secure_malloc【Professional 版のみ】

-head

構築対象のライブラリを指定します。

[指定形式]

```
-head=<sub>[,...]
<sub> : { all | runtime | ctype | math | mathf | stdio | stdlib | string | inttypes }
```

- 省略時解釈
  -head=all 指定と同じです。

[詳細説明]

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

-lang

標準ライブラリの言語規格を指定します。

[指定形式]

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

- 省略時解釈
  -lang=c 指定と同じです。

[詳細説明]

- 標準ライブラリの言語規格を指定します。
- -lang=c を選択すると，C89 規格準拠のものだけで構成し，C99 規格で拡張された関数を含めません。-lang=c99 を選択すると，C89 規格および C99 規格準拠の内容で構成します。
- ライブラリを参照するアプリケーションと，同じ指定にしてください。

[備考]

- C++ の標準ライブラリはサポートしていません。

-secure_malloc【Professional 版のみ】

セキュリティ機能用 malloc 系ライブラリを生成します。

[指定形式]

```
-secure_malloc
```

- 省略時解釈
  通常の malloc 系ライブラリを生成します。

[詳細説明]

- セキュリティ機能用 malloc 系ライブラリを生成します。
- セキュリティ機能用 malloc 系ライブラリを使用する場合，次の操作を行った場合に __heap_chk_fail 関数を呼び出します。
  - calloc，malloc，realloc で割り付けた領域以外のポインタを free，realloc に渡す。
  - free で開放した後のポインタを再度 free，realloc に渡す。
  - calloc，malloc，realloc で割り当てた領域の外側（前後各 2 バイト）に何らかの値を書き込んだ後，割り当てた領域を指すポインタを free，realloc に渡す。
- __heap_chk_fail 関数はユーザが定義する必要があり，動的メモリ管理の異常時に実行する処理を記述します。
- __heap_chk_fail 関数を定義する際には，次の項目に注意してください。
  - __heap_chk_fail 関数は，返却値および引数の型が void 型の far 関数とします。
    void __far __heap_chk_fail(void);
- __heap_chk_fail 関数は，static 関数にしないでください。
- __heap_chk_fail 関数内で再帰的にヒープ・メモリ領域の破壊を検出しないでください。
- セキュリティ機能用の calloc，malloc，および realloc は，領域外への書き込みを検出するために前後各 2 バイト余分に領域を割り当てます。このため，通常より多くヒープ・メモリ領域を消費します。

[注意事項]

- ヒープ・メモリ領域のデフォルト・サイズは 0x100 バイトです。
- ヒープ・メモリ領域を変更する場合は，配列 _REL_sysheap を定義し，配列のサイズを変数 _REL_sizeof_sysheap に設定してください。

```
【ヒープ・メモリ領域設定例】
#include <stddef.h>
#define SIZEOF_HEAP  0x200
int _REL_sysheap[SIZEOF_HEAP / sizeof(int)];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考　　配列 _REL_sysheap は，偶数番地に配置してください。

出力制御

出力制御オプションには，次のものがあります。

- -output

-output

出力ファイルを指定します。

[指定形式]

```
-output=file
```

- 省略時解釈
  カレント・フォルダに stdlib.lib というファイル名で標準ライブラリを生成します。

[詳細説明]

- 標準ライブラリの出力先ファイル名を指定します。
- *file* は絶対パス，または相対パスで指定します。
- *file* を相対パスで指定した場合，カレント・フォルダからの相対パスと解釈します。
- *file* を省略した場合はエラーになります。

その他

その他のオプションには，次のものがあります。

- -logo / -nologo
- -subcommand

-logo / -nologo

コピーライト表示を制御します。

[指定形式]

```
-logo
-nologo
```

- 省略時解釈
  コピーライトを表示します。

[詳細説明]

- -logo 指定時はコピーライトを表示します。-nologo 指定時はコピーライトを表示しません。
- 同時に複数回指定した場合，最後に指定したものが有効になります。

-subcommand

サブコマンド・ファイルを指定します。

[指定形式]

```
-subcommand=file
```

[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。

ライブラリジェネレータに指定できるコンパイル・オプション

ライブラリジェネレータでは，ライブラリ・オプションの他にコンパイル・オプションを指定し，ライブラリをコンパイルするときに用いるオプションとして選択することができます。
ライブラリジェネレータに指定できるコンパイル・オプションは次のものがあります。それ以外のコンパイル・オプションを指定した場合は，エラーになるか，警告を出力せずに無視します。
これらのオプションは，ライブラリを参照するアプリケーション・プログラムと指定を揃えてください。一部のオプションはライブラリ独自の指定が可能であり，「個別指定可否」列に示します。

コンパイル・オプション名	個別指定可否	備考
-cpu	不可	
-use_mda	不可	
-lang	不可	
-dbl_size	不可	
-O[*level]*	可	
-O*item*	可	指定可能な *item* は次の通りです。 - unroll - delete_static_func - inline_level - inline_size - pipeline - tail_call - alias - same_code - branch_chaining - align
-goptimize	可	
-stuff	可	

それぞれのオプションの詳細については「2.5.1 コンパイル・オプション」を参照してください。

2.6 オプションの複数指定

ここでは，ccrl 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。

なお，-subcommand，-asmopt 等を使用して複数のオプションをまとめて指定する場合，以下の注意が必要です。

- -subcommand オプションでファイルを指定した場合，その中に書かれたオプションはコマンド・ライン上の -subcommand を指定した位置に展開され，展開されたオプション，およびその位置に対して本節に記述している規則が適用されます。
- -asmopt，-lnkopt，-asmcmd，-lnkcmd のいずれかを使用してオプションを指定した場合，指定したオプションはコマンド・ラインに展開されないため，それらのオプションには本節に記述している規則は適用されません。

2.6.1 オプションの複数回指定

複数の同じオプションを指定した場合のコンパイラの動作を以下に示します。

1 個だけ指定した場合と同じ動作（パラメータがないもの）	-V，-help，-g，-far_rom，-goptimize，-pass_source，-signed_char，-signed_bitfield，-volatile，-merge_string，-pack，-strict_std，-refs_without_declaration，-large_variable，-nest_comment，-check_language_extension，-Omerge_files，-Ointermodule，-Owhole_program，-g_line，-control_flow_integrity，-unaligned_pointer_for_ca78k0r
すべてのオプションのパラメータが有効	-D，-U，-I，-preinclude，-preprocess，-no_warning_num，-change_message，-subcommand，-asmopt，-asmcmd，-lnkopt，-lnkcmd
あとから指定したオプションとその位置が有効，またはあとから指定したオプションのパラメータが有効	-o，-obj_path，-asm_path，-prep_path，-O*level*，-Oinline_level，-Oinline_size，-Opipeline，-Ounroll，-Odelete_static_func，-Oalias，-Otail_call，-Osame_code，-switch，-character_set，-stack_protector，-stack_protector_all，-lang，-P，-S，-c，-use_mda，-memory_model，-dbl_size，-error_file，-misra2004，-misra2012，-ignore_files_misra，-misra_intermodule，-Obranch_chaining，-Oalign
エラー	-cpu，-dev，-convert_cc

2.6.2 オプションの優先順位

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。
-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の指定は無効となります。ただし，無効となるオプションであっても，オプションに連動するマクロ定義は有効になります。 例 -P と -cpu=S1 を同時に使用する場合，プリプロセス後に処理を終了するため S1 コア用のコード生成は機能しません。しかし，-cpu=S1 に連動する定義済みマクロ __RL78_S1__ は有効となるため，プリプロセス処理で __RL78_S1__ に依存する #ifdef 等の定義は有効になります。
-S	コンパイル処理で終了するため，アセンブル処理以降の処理に関するオプションは無効となります。
-c	アセンブル処理で終了するため，リンカ処理以降の処理に関するオプションは無効となります。
-lang=c99	-misra2004 および -convert_cc は無効となります。

以下のオプションは，ほかのオプションの一部の機能を無効とします。

- -volatile
  外部変数や #pragma address 指定した変数は，-O オプションを指定しても最適化されません。
- -far_rom
  ROM データの near/far 属性は，-memory_model オプションの指定有無にかかわらず，far 属性となります。
- -Oalias
  -Oalias=noansi と -strict_std を同時に指定した場合でも，-Oalias=ansi にはなりません。

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

注 -O*item* は，以下です。
-Ounroll，-Odelete_static_func，-Oinline_level，-Oinline_size，-Opipeline，-Otail_call，
-Osame_code，-Obranch_chaining，-Oalign

2.6.3 機能が矛盾するオプションの組み合わせ

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。

- -dev
  -cpu や -use_mda で指定した内容が -dev で指定したデバイス・ファイルの内容と一致しない場合，エラーとなります。
- -misra2004 と -misra2012
  -misra2004 と -misra2012 を同時に指定した場合，コンパイル・エラーとなります。

2.6.4 オプション間の依存関係

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。

-preprocess	-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，最適化の影響でデバッグ情報がソース行ごとに出ないなどがあります。
-Oinline_level	-merge_files オプションと同時に指定した場合，可能ならファイル間インライン展開を行います。

2.6.5 #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。

- -cpu=S1
  #pragma interrupt，および #pragma interrupt_brk にレジスタ・バンク指定 bank= を使用すると，コンパイル・エラーとなります。
- #pragma 指定した関数や変数を ___near，__far 付きで宣言していない場合，これらの関数や変数の near/far 属性は -cpu，-memmory_model，-far_rom オプション指定の影響を受けます。

2.6.6 near，far との関係

データや関数の near/far 属性は，オプション，およびキーワードで決まります。
near/far 属性の決定方法を以下に示します。

	オプション，キーワード	near/far 属性の決定方法	優先度
(a)	-cpu	デフォルトの near/far 属性を決める。	1
(b)	-memory_model	(a) で決めたデフォルトの near/far 属性を上書きする。	2

	オプション，キーワード	near/far 属性の決定方法	優先度
(c)	-far_rom	(b) で決めた near/far 属性に ROM データのみ far 属性を上書きする。	3
(d)	__near/__far	(a) ～ (c) の影響を受けない。__near，_far 記述が有効になる。	4

上記の表の (b) と (c)，すなわち -memory_model オプションと -far_rom オプションについて，前者だけ指定した場合，および両方を指定した場合の関数，ROM データ，RAM データの near/far 属性を以下に示します。

-memory_model=*type* 指定における *type* の値	-far_rom 指定	関数	ROM データ	RAM データ
small	なし	near	near	near
medium		far	near	near
small	あり	near	far	near
medium		far	far	near

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ファイル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・リスト情報	アセンブラ情報，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・リスト情報	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・リスト情報

アセンブラ情報，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)* RL78 Family Assembler VX.XX.XXx * Assemble Source List *
(2)       (3)                 (4) (5)
OFFSET    CODE                NO  SOURCE STATEMENT

00000000                       1  #CC-RL Compiler RL78 Assembler Source
00000000                       2  #@  CC-RL Version : VX.XX.XXx [DD Mmm YYY]
00000000                       3  #@  Commmand :
00000000                       4  #@   -cpu=S3
00000000                       5  #@   -S
00000000                       6  #@   tp.c
00000000                       7  #@  compiled at Sun May 18 18:59:17 2014
00000000                       8
00000000                       9            .PUBLIC     _label
00000000                      10            .PUBLIC     _func
00000000                      11
00000000                      12            .SECTION    .textf,TEXTF
00000000                      13  _func:
00000000                      14            .STACK      _func = 4
00000000 8F0000               15            mov         a, !LOWW(_label)
00000003 D7                   16            ret
00000000                      17            .SECTION    .bss,BSS
00000000                      18            .ALIGN      2
00000000                      19  _label:
00000000                      20            .DS         2
```

項番	説明
(1)	アセンブラ情報 アセンブラの種別，バージョンを出力します。
(2)	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。

項番	説明
(3)	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。 例 「8F0000」と出力された場合は，下位アドレスから「8F」，「00」，「00」となります。
(4)	行番号 その行の行番号を出力します。インクルード・ファイルの展開も含みます。 10 進数で表記します。
(5)	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。 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。

備考 アセンブル・リスト中の DIVHU 命令と DIVWU 命令の表示

アセンブル・リスト中では，DIVHU 命令と DIVWU 命令を次のように表示します。
すなわち，アセンブラ・ソース中の DIVHU 命令と DIVWU 命令をマクロ展開する形で DIVHU 命令，または DIVWU 命令と NOP 命令に展開します。

例
入力：sample.asm

```
DIVHU   ; comment1
DIVWU   ; comment2
```

出力：sample.prn（抜粋）

```
00000000                1  DIVHU       ; comment1
00000000 CEFB03         2  -- div**
00000003 00             3  -- nop
00000004                4  DIVWU       ; comment2
00000004 CEFB0B         5  -- div**
00000007 00             6  -- nop
```

3.1.3 セクション・リスト情報

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

TEXTF   4 (00000004)        .textf
BSS     2 (00000002)        .bss
```

項番	説明
(1)	セクションの種類 セクションの再配置属性を出力します。
(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
-cpu=S3 tp.asm -prn_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
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	－	出力する
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
	-show=relocation_attribute を指定した場合は，再配置属性を表示します。	-show=relocation_attribute	出力しない
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，および共用体メンバの情報を出力します。	-show=symbol -show=reference -show=struct	出力しない
関数リスト情報	不正な間接関数呼び出し検出で用いる関数リスト情報	-show=cfi	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
ベクタテーブル・アドレス情報	ベクタテーブル・アドレスの内容	-show=vector	出力しない
CRC 情報	CRC の演算結果，および出力位置アドレス	－	-crc オプション指定時は常に出力する

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 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。

3.2.3 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=sub.txt -list -show
```

＜サブコマンド・ファイル sub.txt ＞

```
input sample.obj
```

```
*** Options ***


-subcommand=sub.txt     (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 sub.txt 内で指定したオプションを出力します。

3.2.4 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 ***

** 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。

3.2.5 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

.textf
                                   00000100  0000013b      3c    1
.data
                                   000f0400  000f0403       4    2
.bss
                                   000f0404  000f040b       8    2
```

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。
(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。

-show=relocation_attribute を指定した場合は，セクションに対応している再配置属性を表示します。再配置属性の出力例を以下に示します。

```
*** Mapping List ***

SECTION                            START     END         SIZE    ALIGN   ATTRIBUTE
                                                                           (1)

.textf
                                   00000100  0000013b      3c    1       TEXTF
.data
                                   000f0400  000f0403       4    2       DATA
.bss
                                   000f0404  000f040b       8    2       BSS
```

項番	説明	
(1)	再配置属性 セクションの再配置属性を出力します。 アセンブラの記述に対応して，以下のように出力します。	
	再配置属性 CALLT0 TEXT TEXTF TEXTF_UNIT64KP AT CONST CONSTF DATA DATAF SDATA DATA_AT BSS BSSF SBSS BSS_AT OPT_BYTE SECUR_ID FLASH_SECUR_ID その他	リンクマップ情報表示 CALLT0 TEXT TEXTF TEXTF_UNIT64KP TEXT_AT CONST CONSTF DATA DATAF SDATA DATA_AT BSS BSSF SBSS BSS_AT OPT_BYTE SECUR_ID FLASH_SECUR_ID OTHER

3.2.6 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。

3.2.7 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
                00000100     00000123         24
  _main
                00000100            0         func ,g             0
  _func_01
                00000118            0         func ,g             0
SECTION=.bss
FILE=sample.obj
                000f0404     000f040b         8
  _gvall
                000f0404            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。
(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

項番	説明
(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 オプションを指定した場合は，コンパイル時に -g オプションを指定したソース・ファイル内で定義した構造体，および共用体メンバのアドレスを出力します。
シンボル情報の出力例を以下に示します。

```
*** Symbol List ***

SECTION
FILE
                START         END             SIZE
  SYMBOL        ADDR          SIZE            INFO          COUNTS  OPT
    (1)                       (2)
  STRUCT                      SIZE
    (3)         (4)           (5)             (6)
  MEMBER        ADDR          SIZE            INFO

SECTION=B
FILE=sample.obj
                00001000      00001003        4
  a
                00001000      4               data ,g       1
  struct A{
                              4
  a.b
                00001000      1               char
  a.c
                00001002      2               short
```

項番	説明
(1)	型名 構造体，および共用体の型名を出力します。
(2)	サイズ 構造体，および共用体のサイズを出力します。
(3)	メンバ名 構造体，および共用体のメンバ名を出力します。
(4)	メンバのアドレス 構造体，および共用体のメンバのアドレスを出力します。
(5)	メンバのサイズ 構造体，および共用体のメンバのサイズを出力します。 ビット・フィールドの場合も，構造体，および共用体のメンバの型サイズを出力します。

項番	説明
(6)	メンバの型名 構造体，および共用体のメンバの型名を出力します。 ビット・フィールドの場合も，構造体，および共用体のメンバの型名を出力します。 ポインタ型の場合は以下のように出力します。 [pointer] : __near/__far の指定がない場合 [near pointer] : near ポインタと指定されている場合 [far pointer] : far ポインタと指定されている場合

3.2.8 関数リスト情報

-show=cfi を指定した場合，不正な間接関数呼び出し検出で用いる関数リストの内容を出力します。
出力例を以下に示します。

```
*** CFI Table List ***

SYMBOL/ADDRESS

_func     (1)
0000F100  (2)
```

項番	説明
(1)	関数シンボルを出力します。
(2)	関数シンボルが定義されていない場合，関数アドレスを出力します。

3.2.9 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
                               00000100
                _func_01
                               00000118
     SECTION=.data
                _gval3
                               000f0400  0003(00000032:.text)
                                         0003(00000038:.text)
     SECTION=.bss
                _gval1
                               000f0404  0001(0000001a:.text)
                                         0001(00000020:.text)
                _gval2
                               000f0408  0002(00000026:.text)
                                         0002(0000002c:.text)
0002 sample2
     SECTION=.text
                _func02
                               00000124  0001(0000000a:.text)
0003 sample3
     SECTION=.text
                _func03
                               00000130  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)

3.2.10 ベクタテーブル・アドレス情報

-show=vector オプションを指定した場合，ベクタテーブル・アドレスの内容を出力します。
ベクタテーブル・アドレス情報の出力例を以下に示します。

```
*** Variable Vector Table List ***
(1)     (2)
ADDRESS SYMBOL/ADDRESS
00      start
02      dummy
04      INTWDTI
06      0000F100
 :
```

項番	説明
(1)	ベクタテーブル・アドレス ベクタテーブル・アドレスを出力します。
(2)	シンボル シンボルを出力します。 シンボルが定義されていない場合はアドレスで出力します。

3.2.11 CRC 情報

-crc オプションを指定した場合，CRC の演算結果，および出力位置アドレスを出力します。
CRC 情報の出力例を以下に示します。

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

項番	説明
(1)	CRC の演算結果 CRC の演算結果を出力します。
(2)	CRC の演算結果の出力位置アドレス CRC の演算結果の出力位置アドレスを出力します。

3.3 リンク・マップ・ファイル（オブジェクト結合時）

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・ファイルの場合に最適化リンカが出力するリンク・マップ・ファイルの内容と形式について説明します。

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

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。
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=sub.txt -list
```

＜サブコマンド・ファイル sub.txt ＞

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

```
*** Options ***

-subcommand=sub.txt     (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 sub.txt 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。

項番	説明
(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=sub.txt -list -show
```

＜サブコマンド・ファイル sub.txt ＞

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

```
*** Options ***

-subcommand=sub.txt     (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 sub.txt 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=RL78                (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。

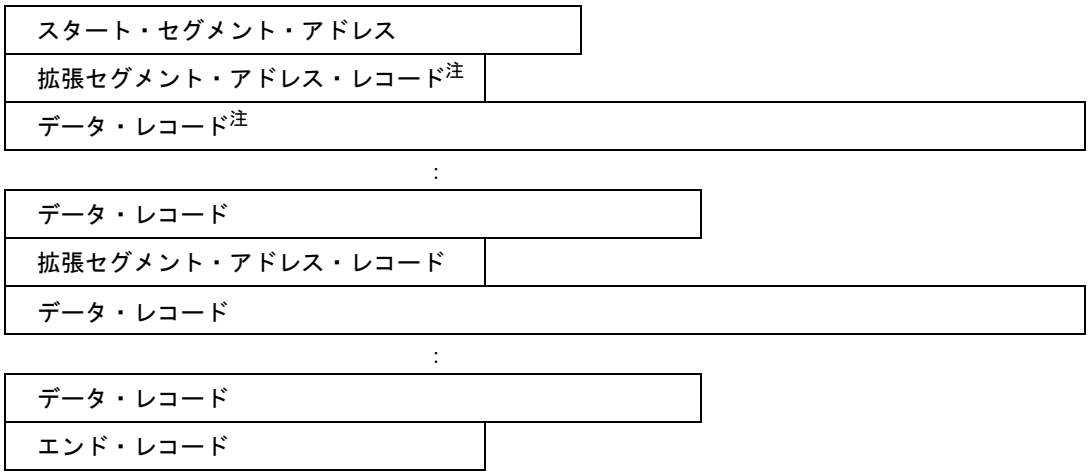
3.5.1 インテル拡張ヘキサ・ファイルの構成

インテル拡張ヘキサ・ファイル（20 ビット）は，スタート・セグメント・アドレス・レコード，拡張セグメント・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・アドレス・レコード，スタート・セグメント・アドレス，拡張セグメント・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　　拡張セグメント・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・セグメント・アドレス・レコード	エントリ・ポイント・アドレス
拡張セグメント・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・セグメント・アドレス 02 ： 拡張セグメント・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・セグメント・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 拡張セグメント・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

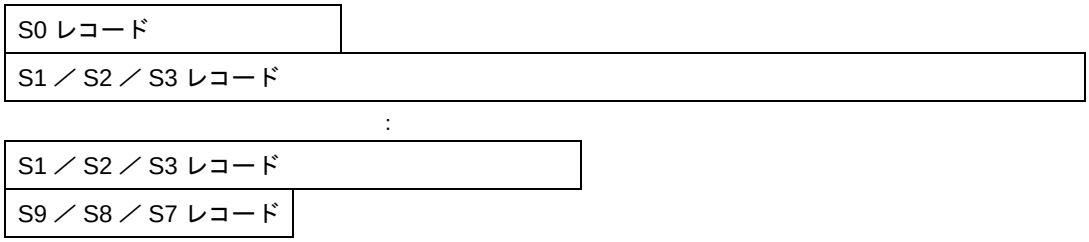
モトローラ・S タイプ・ファイルは，ヘッダ・レコードである S0 レコード，データ・レコードである S1 ／ S2 ／ S3 レコード，エンド・レコードである S9 ／ S8 ／ S7 レコードの 7 種類のレコード[注 1] により構成されます[注 2]。

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

項番	説明
(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

3.7 変数 / 関数情報ファイル

ここでは，変数 / 関数情報ファイルについて説明します。
変数 / 関数情報ファイルは，C ソース・ファイル中で定義された変数，および関数に対して，saddr 変数や callt 関数の宣言を記述したテキスト形式のファイルです。

3.7.1 変数 / 関数情報ファイルの出力

- -lnkopt=-vfinfo オプションを指定すると，最適化リンカが変数 / 関数情報ファイルを出力します。
  -vfinfo オプションについての詳細は，「-VFINFO」を参照してください。
- 最適化リンカは，変数のサイズ，および変数や関数の参照頻度を元にして，コードサイズの削減効果が高い変数や関数を選択し，それらの変数や関数に対して #pragma 指令による saddr 変数，callt 関数，near 関数の宣言を追加したヘッダ・ファイル（変数 / 関数情報ファイル）を出力します。
  #pragma saddr については「saddr 領域利用（#pragma saddr）」，#pragma callt については「callt 関数（#pragma callt）」，#pragma near については「near/far 関数（#pragma near/#pragma far）【V1.05 以降】」を参照してください。
- 以下の変数や関数は，変数 / 関数情報ファイルの対象にはなりません。
  ただし，割り込みハンドラについては，コメントアウトして変数 / 関数情報ファイルに表示されます。
  - 標準ライブラリ関数，ランタイムライブラリ関数
  - ソフトウェア割り込みハンドラ，ハードウェア割り込みハンドラ，RTOS 割り込みハンドラ
  - アセンブリ・ソースで定義した変数や関数
- 変数 / 関数情報ファイルの出力例を以下に示します。

```
/* RENESAS OPTIMIZING LINKER GENERATED FILE 2014.10.20 */
/*** variable information ***/
(1)                  (2)       (3)   (4)   (5)
#pragma saddr var1 /* count:10,size:1,near, file1.obj */
(6)                        (2)      (3)    (4)  (7)    (5)
/* #pragma saddr var2 */ /* count: 0,size:2,near,unref, file2.obj */

/*** function information ***/
(8)                    (9)      (10)  (11)
#pragma callt func1 /* count:20,near, file1.obj */
(12)                   (9)      (10) (11)
#pragma near func2 /* count:10,far, file2.obj */
(13)                       (9)      (10) (14)  (11)
/* #pragma near func3 */ /* count: 0,far,unref, file3.obj */
```

項番	説明
(1)	変数情報 #pragma 指令による saddr 変数の宣言を出力します。
(2)	参照回数 変数の参照回数を出力します。
(3)	変数のサイズ 変数のサイズを出力します。
(4)	参照タイプ 変数の元の参照タイプを near，far，saddr で出力します。
(5)	ファイル名 変数が所属するオブジェクトファイル名を出力します。
(6)	変数情報 saddr 領域から溢れた変数をコメントで出力します。

項番	説明
(7)	変数の補足情報 変数の補足情報を出力します。補足情報は次のものがあります。 unref : 変数の参照がない場合に出力します。 const : const 変数の場合に出力します。 fix : 変数の配置がリロケータブルでない場合に出力します。 unrecognizable : 変数をシンボルとして認識できない場合に出力します。
(8)	関数情報 #pragma 指令による callt 関数の宣言を出力します。
(9)	参照回数 関数の参照回数を出力します。
(10)	関数の呼び出し方法 関数の元の呼び出し方法を near，far，callt で出力します。
(11)	ファイル名 関数が所属するオブジェクトファイル名を出力します。
(12)	関数情報 #pragma 指令による near 関数の宣言を出力します。
(13)	関数情報 callt，near 領域から溢れた関数をコメントで出力します。
(14)	関数の補足情報 関数の補足情報を出力します。補足情報は次のものがあります。 unref : 関数の参照がない場合に出力します。 interrupt : 割り込みハンドラの場合に出力します。 unrecognizable : 関数をシンボルとして認識できない場合に出力します。

3.7.2 変数 / 関数情報ファイルの利用

- 変数 / 関数情報ファイルを C ソース・ファイルからインクルードしてコンパイルしてください。
  コードサイズが削減されたオブジェクト・コードが得られます。
- あるいは，変数 / 関数情報ファイルををインクルードするのに -preinclude オプションを指定してください。
  この方法を使うと，C ソース・ファイルを書き換える必要がありません。
- 変数 / 関数情報ファイルは編集することができます。
  これにより，変数 / 関数情報ファイル中の大域変数に対する saddr 指定や関数に対する callt/near 指定の有効・無効化と C ソース・ファイル中の static 変数に対する saddr 指定や static 関数に対する callt/near 指定の追加・削除を通してコードサイズをチューニングすることができます。

4. コンパイラ言語仕様

この章では，CC-RL 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

本節では，C90 および C99 規格に対する CC-RL での処理系固有の定義について説明します。
なお，CC-RL 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 C90 の処理系定義

この項では，C90 規格における処理系定義について説明します。

(1) どのような方法で診断メッセージを識別するか。(5.1.1.3)
「10. メッセージ」を参照してください。

(2) main 関数への実引数の意味。(5.1.2.2.1)
フリースタンディング環境であるため，規定しません。

(3) 対話型装置がどのようなもので構成されるか。(5.1.2.3)
対話型装置の構成については，特に規定しません。

(4) 外部給合でない識別子において（31 以上の）意味がある先頭の文字数。(6.1.2)
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。

(5) 外部結合である識別子において（6 以上の）意味がある先頭の文字数。(6.1.2)
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。

(6) 外部結合である識別子において英小文字と英大文字の区別に意味があるか否か。(6.1.2)
識別子内の英小文字と英大文字を区別します。

(7) ソース及び実行文字集合の要素で，この規格で明示的に規定しているもの以外の要素。(5.2.1)
ソースおよび実行文字集合の要素の値は，ASCII コード，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) 多バイト文字のコード化のために使用されるシフト状態。(5.2.1.2)
シフト状態はサポートしていません。

(9) 実行文字集合の文字におけるビット数。(5.2.4.2.1)
8 ビットとします。

(10) （文字定数内及び文字列リテラル内の）ソース文字集合の要素と実行文字集合の要素との対応付け。(6.1.3.4)
同一の値をもつ要素へ対応付けます。

(11) 基本実行文字集合で表現できない文字若しくは拡張表記を含む単純文字定数の値，又はワイド文字定数に対しては拡張文字集合で表現できない文字若しくは拡張表記を含むワイド文字定数の値。(6.1.3.4)
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。

拡張表記	値（ASCII）
¥a	0x07
¥b	0x08
¥f	0x0C
¥n	0x0A
¥r	0x0D
¥t	0x09
¥v	0x0B

(12) 2 文字以上の文字を含む単純文字定数又は 2 文字以上の多バイト文字を含むワイド文字定数の値。(6.1.3.4)
2 文字までの文字を含む単純文字定数は，末尾の文字を下位バイト，先頭の文字を上位バイトに持つ 2 バイトの値を持ちます。3 文字以上の文字を持つ文字定数はエラーとなります。基本的な実行環境文字集合で表現されない文字は，その値を持つ単純文字定数とみなします。不正な逆斜線表記は逆斜線を無視して次の文字を単純文字定数とみなします。

(13) ワイド文字定数に対して，多バイト文字を対応するワイド文字（コード）に変換するために使用されるロケール。(6.1.3.4)
ロケールはサポートしていません。

(14) " 単なる "char が signed char と同じ値の範囲をもつか，unsigned char と同じ値の範囲をもつか。(6.2.1.1)
char 型は，unsigned char 型と同じ値の範囲，同じ表現形式，同じ動作を持ちます。ただし，オプション -signed_char で signed char 型に切り替えが可能です。

(15) 整数の様々な型の表現方法及び値の集合。(6.1.2.5)
「4.1.3 データの内部表現と領域」を参照してください。

(16) 整数をより短い符号付き整数に変換した結果，又は符号無し整数を長さの等しい符号付き整数に変換した結果で，値が表現できない場合の変換結果。(6.2.1.2)
変換先の型の幅でマスクした（上位ビットを切り捨てた）ビット列とします。

(17) 符号付き整数に対してビット単位の演算を行った結果。(6.3)
シフト演算子の場合は算術シフトを行います。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。

(18) 整数除算における剰余の符号。(6.3.5)
"%" 演算子の結果の符号は第 1 オペランドの符号とします。

(19) 負の値をもつ符号付き汎整数型の右シフトの結果。(6.3.7)
算術シフトを行います。

(20) 浮動小数点数の様々な型の表現方法及び値の集合。(6.1.2.5)
「4.1.3 データの内部表現と領域」を参照してください。

(21) 汎整数の値を元の値に正確に表現することができない浮動小数点数に変換する場合の切捨ての方向。(6.2.1.3)
表現しうる最も近い方向に丸められます。

(22) 浮動小数点数をより狭い浮動小数点数に変換する場合の切捨て又は丸めの方向。(6.2.1.4)
表現しうる最も近い方向に丸められます。

(23) 配列の大きさの最大値を保持するために必要な整数の型。すなわち sizeof 演算子の型 size_t。(6.3.3.4，7.1.1)
unsigned int 型です。

(24) ポインタを整数へキャストした結果，及びその逆の場合の結果。(6.3.4)
「4.2.6 拡張言語仕様の使用方法」にある「メモリ配置領域指定（__near/__far）」を参照してください。

(25) 同じ配列内の，二つの要素へのポインタ間の差を保持するために必要な整数の型，すなわち ptrdiff_t の型。(6.3.4，7.1.1)
signed int 型です。

(26) register 記憶域クラス指定子を使用することによって実際にオブジェクトをレジスタに置くことができる範囲。(6.5.1)
register 指定子を無視して最適化します。

(27) 共用体オブジェクトのメンバを異なる型のメンバを用いてアクセスする場合。(6.3.2.3)
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，共用体オブジェクトのメンバを異なる型のメンバを用いてアクセスする場合，データの内部表現はアクセスする型に従います。

(28) 構造体のメンバの詰め物及び境界調整。(6.5.2.1)
「4.1.3 データの内部表現と領域」を参照してください。

(29) " 単なる "int 型のビットフィールドが，signed int のビットフィールドとして扱われるか，unsigned int のビットフィールドとして扱われるか。(6.5.2.1)
unsigned int 型として扱います。ただし，オプション -signed_bitfield により変更可能です。

(30) 単位内のビットフィールドの割付け順序。(6.5.2.1)
下位から割り付けます。

(31) ビットフィールドを記憶域単位の境界にまたがって割り付けうるか否か。(6.5.2.1)
構造体パッキング未指定時は，ビットフィールドは境界を跨がず，次の領域に割り付けます。構造体パッキング指定時は，ビットフィールドは境界を跨ぐことがあります。

(32) 列挙型の値を表現するために選択される整数型。(6.5.2.2)
char，signed char，unsigned char，signed short 型のいずれかです。列挙値が収まる最小の型となります。

(33) volatile 修飾型のオブジェクトへのアクセスをどのように構成するか。(6.5.3)
アクセス幅，アクセス順序，アクセス回数は C ソース上の記述通りに実施しますが，対応するマイコンの命令がない型へのアクセスは，その限りではありません。

(34) 算術型，構造体型又は共用体型を修飾する宣言子の最大数。(6.5.4)
128 です。

(35) switch 文における case 値の最大数。(6.6.4.2)
65535 です。

(36) 条件付き取込みを制御する定数式中の単一文字からなる文字定数の値が実行文字集合中の同じ文字定数の値に一致するか否か。このような文字定数が負の値をもつことがあるか否か。(6.8.1)
条件付き取り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。
単なる char 型（signed も unsigned もつかない char）が unsigned の場合は，負の値を持ちません。signed の場合は，負の値を持つことがあります。

(37) 取込み可能なソース・ファイルを捜すための方法。(6.8.2)
次の順序で探索し，フォルダにある同名ファイルをヘッダと識別します。
(1) フルパス指定の場合はそのパスが示すフォルダ
(2) オプション -I で指定されたフォルダ
(3) 標準インクルードファイルフォルダ（コンパイラが置かれた bin フォルダからの相対パスでの ..¥inc フォルダ）

(38) 取込み可能なソース・ファイルに対する " で囲まれた名前の探索。(6.8.2)
次の順序で探索します。
(1) フルパス指定の場合はそのパスが示すフォルダ
(2) ソース・ファイルがあるフォルダ
(3) オプション -I で指定されたフォルダ
(4) 標準インクルードファイルフォルダ（コンパイラが置かれた bin フォルダからの相対パスでの ..¥inc フォルダ）

(39) ソース・ファイル名と文字列との対応付け。(6.8.2)
#include に記述された文字列は，ソース文字集合として指定した文字コードとして解釈され，ヘッダ名または外部ソース・ファイル名に対応付けられます。

(40) 認識される #pragma 指令の動作。(6.8.6)
「4.2.4 #pragma 指令」を参照してください。

(41) 翻訳日付及び翻訳時刻がそれぞれ有効でない場合における __DATE__ 及び __TIME__ の定義。(6.8.8)
日付や時刻が得られない場合はありません。

(42) マクロ NULL が展開する空ポインタ定数。(7.1.6)
(void*)0 とします。

(43) assert 関数によって表示される診断メッセージ及び assert 関数の終了時の動作。(7.2)
以下の通りです。
Assertion failed : *式* , file *ファイル名* , line *行番号*
終了時の動作は，abort 関数の実装に依存します。

(44) isalnum 関数，isalpha 関数，iscntrl 関数，islower 関数，isprint 関数及び isupper 関数によってテストされる文字集合。(7.3.1)
unsigned char 型（0 ～ 255）および EOF（-1）です。

(45) 数学関数に定義域エラーが発生した場合に返される値。(7.5.1)
「7.5 ライブラリ関数」を参照してください。

(46) アンダフロー値域エラーの場合に，数学関数がマクロ ERANGE の値を整数式 errno に設定するか否か。(7.5.1)
アンダーフロー発生時は errno に ERANGE を設定します。

(47) fmod 関数の第 2 引数が 0 の場合に，定義域エラーが発生するか又は 0 が返されるか。(7.5.6.4)
定義域エラーが発生します。詳細は fmod 関数群の説明を参照してください。

(48) signal 関数に対するシグナルの集合。(7.7.1.1)
シグナル操作関数はサポートしていません。

(49) signal 関数によって認識されるシグナルの意味。(7.7.1.1)
シグナル操作関数はサポートしていません。

(50) signal 関数によって認識されるシグナルに対する既定の処理及びプログラム開始時の処理。(7.7.1.1)
シグナル操作関数はサポートしていません。

(51) シグナル処理ルーチンの呼出しの前に signal(sig, SIG_DFL); と同等のことが実行されない場合のシグナルの遮断の処置。(7.7.1.1)
シグナル操作関数はサポートしていません。

(52) シグナル関数によって指定された処理ルーチンによって SIGILL シグナルが受け付けられる場合に既定の処理が再設定されるか否か。(7.7.1.1)
シグナル操作関数はサポートしていません。

(53) テキストストリームの最終行が，終了を示す改行文字を必要とするか否か。(7.9.2)
改行文字を必要としません。

(54) データが読み取られる時，改行文字の直前にテキストストリームに書き込まれた空白文字の並びが現れるか否か。(7.9.2)
データが読み取られる時に現れます。

(55) バイナリストリームに書き込むデータに付加してもよいナル文字の個数。(7.9.2)
0 個です。

(56) 追加モードのストリームのファイル位置表示子が，最初にファイルの先頭又は終わりのどちらに位置付けされるか。(7.9.3)
ファイル操作関数はサポートしていません。

(57) テキストストリームへの書込みが，結び付けられたファイルを最終書込み点の直後で切り捨てるか否か。(7.9.3)
ファイル操作関数はサポートしていません。

(58) ファイルバッファリングの特性。(7.9.3)
ファイル操作関数はサポートしていません。

(59) 長さ 0 のファイルが実際に存在するか否か。(7.9.3)
ファイル操作関数はサポートしていません。

(60) 正しいファイル名の規則。(7.9.3)
ファイル操作関数はサポートしていません。

(61) 同一ファイルを複数回オープンすることが可能か否か。(7.9.3)
ファイル操作関数はサポートしていません。

(62) オープンされているファイルに対する remove 関数の効果。(7.9.4.1)
ファイル操作関数はサポートしていません。

(63) rename 関数呼出し前に新しい名前をもつファイルが存在している場合の効果。(7.9.4.2)
ファイル操作関数はサポートしていません。

(64) fprintf 関数中の %p 変換による出力。(7.9.6.1)
16 進表記です。
fprintf 関数はサポートしていません。

(65) fscanf 関数中の %p 変換に対する入力。(7.9.6.2)
16 進整数です。
fscanf 関数はサポートしていません。

(66) fscanf 関数中の %[変換において文字 - が走査文字の並び中の最初の文字でも最後の文字でもない場合の解釈。(7.9.6.2)
「7.5.7 標準入出力関数」にある「scanf」を参照してください。
fscanf 関数はサポートしていません。

(67) fgetpos 関数又は ftell 関数が失敗した場合に，マクロ errno に設定される値。(7.9.9.1，7.9.9.4)
ファイル操作関数はサポートしていません。

(68) perror 関数によって生成されるメッセージ。(7.9.10.4)
「7.5 ライブラリ関数」を参照してください。

(69) 要求された大きさが 0 の場合の calloc 関数，malloc 関数又は realloc 関数の動作。(7.10.3)
NULL を返します。

(70) オープンされているファイル及び一時ファイルに関しての abort 関数の動作。(7.10.4.1)
ファイル操作関数はサポートしていません。

(71) exit 関数の実引数の値が 0，EXIT_SUCCESS 又は EXIT_FAILURE 以外の場合に返される状態。(7.10.4.3)
フリースタンディング環境であるため，規定しません。

(72) getenv 関数によって使われる環境の名前の集合及び環境の並びを変更する方法。(7.10.4.4)
getenv 関数はサポートしていません。

(73) system 関数による文字列の実行のモード及び内容。(7.10.4.5)
system 関数はサポートしていません。

(74) strerror 関数によって返されるエラーメッセージ文字列の内容。(7.11.6.2)
「7.5 ライブラリ関数」を参照してください。

(75) 地方時及び夏時間。(7.12.1)
time.h はサポートしていません。

(76) clock 関数のための時点。(7.12.2.1)
time.h はサポートしていません。

翻訳限界

CC-RL の翻訳限界を以下に示します。
「制限なし」の項目は，上限はホスト環境のメモリ状況に依存します。

表 4.1 翻訳限界（C90）

項目	C90	CC-RL
条件付き取込みにおける入れ子のレベル数	8	制限なし
一つの宣言中の一つの算術型，構造体型，共用体型又は不完全型を修飾するポインタ，配列及び関数宣言子（の任意の組合せ）の個数	12	128
一つの完全宣言子における括弧で囲まれた宣言子の入れ子のレベル数	31	制限なし
一つの完全式における括弧で囲まれた式の入れ子のレベル数	32	制限なし
内部識別子又はマクロ名において意味がある先頭の文字数	31	制限なし
外部識別子において意味がある先頭の文字数	6	制限なし
一つの翻訳単位中における外部識別子数	511	制限なし
一つのブロック中におけるブロック有効範囲をもつ識別子数	127	制限なし
一つの翻訳単位中で同時に定義されうるマクロ識別子数	1024	制限なし
一つの関数定義における仮引数の個数	31	制限なし
一つの関数呼出しにおける実引数の個数	31	制限なし
一つのマクロ定義における仮引数の個数	31	制限なし
一つのマクロ呼出しにおける実引数の個数	31	制限なし
一つの論理ソース行における文字数	509	制限なし
（連結後の）単純文字列リテラル又はワイド文字列リテラル中における文字数	509	制限なし
（ホスト環境の場合）一つのオブジェクトのバイト数	32767	32767（65535）注 1
#include で取り込まれるファイルの入れ子のレベル数	8	制限なし
一つの switch 文（入れ子になった switch 文を除く）中における case 名札の個数	257	65535
一つの構造体又は共用体のメンバ数	127	制限なし
一つの列挙体における列挙定数の個数	127	制限なし
一つのメンバ宣言並びにおける構造体又は共用体定義の入れ子のレベル数	15	制限なし

注 1. カッコ内は -large_variable 指定時

4.1.2 C99 の処理系定義

この項では，C99 規格における処理系定義について説明します。

(1) 翻訳時の構文規則違反等に対する診断メッセージの出し方。（3.10, 5.1.1.3）
「10. メッセージ」を参照してください。

(2) 翻訳フェーズ 3 において，改行文字を除く空白類文字の並びを保持するか一つの空白文字に置き換えるか。（5.1.1.2）
そのまま保持されます。

(3) 翻訳フェーズ 1 での，物理的なソース・ファイルの多バイト文字と対応するソース文字集合のマッピング方法。（5.1.1.2）
多バイト文字は，コンパイルオプションにより対応するソース文字集合にマッピングします。

(4) フリースタンディング環境におけるプログラム開始時に呼び出される関数の名前と型。（5.1.2.1）
規定しません。スタート・アップの実装に依存します。

(5) フリースタンディング環境におけるプログラム終了処理の効果。（5.1.2.1）
正常終了時はスタート・アップに依存します。プログラムを異常終了させる場合は abort 関数を使用します。

(6) main 関数を定義できる代替方法。（5.1.2.2.1）
フリースタンディング環境であるため，規定しません。

(7) main 関数の argv 実引数が指す文字列の値。（5.1.2.2.1）
フリースタンディング環境であるため，規定しません。

(8) ユーザーインタフェースとなる対話型装置がどのようなものであるか。（5.1.2.3）
対話型装置の構成については，特に規定しません。

(9) シグナル全体の集合，それぞれの意味および既定の操作。（7.14）
シグナル操作関数はサポートしていません。

(10) SIGFPE，SIGILL，SIGSEGV 以外の，計算例外に対応するシグナルの値。（7.14.1.1）
シグナル操作関数はサポートしていません。

(11) プログラム開始時に実行される，signal(sig, SIG_IGN); と同等なシグナル。（7.14.1.1）
シグナル操作関数はサポートしていません。

(12) getenv 関数で使用される，環境の並びに定義されている名前の集合および環境の並びを変更する方法。（7.20.4.5）
getenv 関数はサポートしていません。

(13) system 関数における，引数 string が指す文字列の実行方法。（7.20.4.6）
system 関数はサポートしていません。

(14) ソース基本文字集合の一部でない多バイト文字が識別子の中に現れることを許すかどうか，及び識別子に使用してよい多バイト文字およびその文字と国際文字名との対応。（6.4.2）
識別子として多バイト文字は使用できません。

(15) 識別子における，意味がある先頭の文字の個数。（5.2.4.1，6.4.2）
識別子すべてが意味のあるものとして扱われます。また，識別子の長さは無制限です。

(16) 1 バイトあたりのビット数。（3.6）
8 ビットとします。

(17) 実行文字集合の要素の値。（5.2.1）
実行文字集合の要素の値は，ASCII コード，EUC，SJIS，UTF-8，big5，gb2312 の値です。

(18) 標準の英字逆斜線表記のそれぞれに割り当てられた実行文字集合の要素の一意な値。（5.2.2）

逆斜線表記	値（ASCII）
"¥a"	0x07
"¥b"	0x08
"¥f"	0x0C
"¥n"	0x0A
"¥r"	0x0D
"¥t"	0x09
"¥v"	0x0B

(19) 実行基本文字集合の任意の要素以外の文字が格納された char 型のオブジェクトの値。（6.2.5）
char 型に型変換した値となります。

(20) signed char と unsigned char のどちらが，単なる char と同じ値の範囲，同じ表現形式，同じ動作を持つのか。（6.2.5, 6.3.1.1）
char 型は，unsigned char 型と同じ値の範囲，同じ表現形式，同じ動作を持ちます。ただし，オプション -signed_char で signed char 型に切り替えが可能です。

(21) 文字定数と文字列リテラル中のソース文字集合の要素から実行文字集合の要素への対応付け方法。（6.4.4.4, 5.1.1.2）
同一の値をもつ要素へ対応付けます。

(22) 2 文字以上を含む，または 1 バイトの実行文字で表現できない文字もしくは逆斜線表記を含む単純文字定数の値。（6.4.4.4）
2 文字までの文字を含む単純文字定数は，末尾の文字を下位バイト，先頭の文字を上位バイトに持つ 2 バイトの値を持ちます。3 文字以上の文字を持つ文字定数はエラーとなります。基本的な実行環境文字集合で表現されない文字は，その値を持つ単純文字定数とみなします。不正な逆斜線表記は逆斜線を無視して次の文字を単純文字定数とみなします。

(23) 2 文字以上の多バイト文字を含む，または実行拡張文字集合で表現できない多バイト文字もしくは逆斜線表記を含むワイド文字定数の値。（6.4.4.4）
多バイト文字としての左端 1 文字の値となります。

(24) 実行拡張文字集合の 1 つの文字に対応する単一の多バイト文字を含むワイド文字定数の値をワイド文字に対応させる，その時点のロケール。（6.4.4.4）
ロケールはサポートしていません。

(25) ワイド文字列リテラルから対応するワイド文字コードへの変換時に使用される，その時点のロケール。（6.4.5）
ロケールはサポートしていません。

(26) 実行文字集合で表現できない多バイト文字もしくは逆斜線表記を含む文字列リテラルの値。（6.4.5）
逆斜線表記は対応するバイトの値，多バイト文字はそれぞれのバイトの値になります。

(27) 処理系が提供する拡張整数型。（6.2.5）
拡張整数型は提供していません。

(28) 符号付き整数型が符号と絶対値，2 の補数，または 1 の補数のいずれの表現方式を使用して表現されるか，および異常値がトラップ表現か通常値かどうか。（6.2.6.2）
符号付き整数型は 2 の補数で表現します。トラップ表現はありません。

(29) 同じ精度を持つ別の拡張整数型に対する，整数拡張型の順位。（6.3.1.1）
拡張整数型は提供していません。

(30) 整数型の値を符号付き整数型に変換する際，値が変換先の型で表現できない場合の結果，あるいは生成されるシグナル。（6.3.1.3）
変換先の型の幅でマスクした（上位ビットを切り捨てた）ビット列とします。

(31) 符号付き整数に対するビット単位の操作の結果。（6.5）
シフト演算子の場合は算術シフトを行います。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。

(32) 浮動小数点演算の正確度，<math.h> および <complex.h> の中で定義される，浮動小数点型の結果を返却する，ライブラリ関数の正確度。（5.2.4.2.2）
不明です。

(33) FLT_ROUNDS に対する非標準の値において，特徴付けられた丸め動作。（5.2.4.2.2）
FLT_ROUNDS に対する非標準の値は定義しません。

(34) FLT_EVAL_METHOD に対する非標準の負の値によって特徴付けられる評価形式。（5.2.4.2.2）
FLT_EVAL_METHOD に対する非標準の値は定義しません。

(35) 整数が，元の値を正確に表現できない浮動小数点数に変換された時の丸めの方向。（6.3.1.4）
表現しうる最も近い方向に丸められます。

(36) 浮動小数点数がより狭い浮動小数点数型に変換された時の丸めの方向。（6.3.1.5）
表現しうる最も近い方向に丸められます。

(37) 浮動小数点定数を，最も近い表現可能な値とするか，それとも最も近い表現可能な値のすぐ隣（大きい値もしくは小さい値）で表現可能な値とするか。（6.4.4.2）
最近値に丸めます。

(38) FP_CONTRACT プラグマがない場合，式が短縮されるかどうか。短縮されるならどのように短縮されるか。（6.5）
式の短縮は，各オプション指定に依存します。
FP_CONTRACT プラグマは機能しません。
#pragma STDC FP_CONTRACT 指定をしても無視します。

(39) FENV_ACCESS プラグマの既定の状態。（7.6.1）
FENV_ACCESS プラグマの既定の状態は ON になります。
ただし，#pragma STDC FENV_ACCESS 指定しても無視します。

(40) 付加的な浮動小数点例外，丸めモード，環境，分類，およびそれらのマクロ名。（7.6, 7.12）
コンパイラが提供しているライブラリ math.h に準じます。付加的な定義はありません。

(41) FP_CONTRACT プラグマの既定の状態。（7.12.2）
FP_CONTRACT プラグマの既定の状態は ON になります。

(42) IEC 60559 に準拠した処理系で，丸めの結果が実際に数学的な結果と同等である時に，" 不正確結果 " 浮動小数点例外が生成されうるかどうか。（F.9）
浮動小数点例外をサポートしていません。
" 不正確結果 " 浮動小数点例外は生成されません。

(43) IEC 60559 に準拠した処理系で，結果が極めて小さいが不正確ではない時に，" アンダーフロー" 浮動小数点例外や " 不正確結果 " 浮動小数点例外が生成されうるかどうか。（F.9）
浮動小数点例外をサポートしていません。" アンダーフロー " 浮動小数点例外や " 不正確結果 " 浮動小数点例外は生成されません。

(44) ポインタから整数への変換の結果，またはその逆の結果。（6.3.2.3）
「4.2.6 拡張言語仕様の使用方法」にある「メモリ配置領域指定（__near/__far）」を参照してください。

(45) 同じ配列の要素への 2 つのポインタの減算の結果の大きさ。（6.5.6）
結果の型は signed int 型となります。

(46) register 記憶域クラス指定子の使用がどのくらい効果を持つか。（6.7.1）
register 指定子を無視して最適化します。

(47) inline 関数指定子の使用がどのくらい効果を持つか。（6.7.4）
常に展開を試みます。ただし，条件によっては展開を行わないことがあります。

(48) 単なる int ビットフィールドが signed int ビットフィールドとして扱われるか，それとも unsigned int ビットフィールドとして扱われるか。（6.7.2，6.7.2.1）
unsigned int 型として扱います。ただし，オプション -signed_bitfield により変更可能です。

(49) _Bool，signed int，unsigned int 以外で許されるビットフィールドの型。（6.7.2.1）
全ての整数型が許されています。

(50) ビットフィールドが記憶域単位の境界を跨ぐかどうか。（6.7.2.1）
構造体パッキング未指定時は，ビットフィールドは境界を跨がず，次の領域に割り付けます。構造体パッキング指定時は，ビットフィールドは境界を跨ぐことがあります。

(51) 記憶域単位内のビットフィールド割付けの順序。（6.7.2.1）
下位から割り付けます。

(52) 構造体または共用体オブジェクトのビットフィールド以外の各メンバの境界の調整方法。（6.7.2.1）
「4.1.3 データの内部表現と領域」を参照してください。

(53) それぞれの列挙型が適合する整数型。（6.7.2.2）
char，signed char，unsigned char，signed short 型のいずれかです。列挙値が収まる最小の型となります。

(54) volatile 修飾型のオブジェクトへのアクセスをどのように構成するか。（6.7.3）
アクセス幅，アクセス順序，アクセス回数は C ソース上の記述通りに実施しますが，対応するマイコンの命令がない型へのアクセスは，その限りではありません。

(55) #include のヘッダ名の 2 つの形式中の文字列が，ヘッダまたは外部ソース・ファイルの名前に対応付けられる方法。（6.4.7）
#include に記述された文字列は，ソース文字集合として指定した文字コードとして解釈され，ヘッダ名または外部ソース・ファイル名に対応付けられます。

(56) 条件付き取り込みを制御する定数式中の文字定数の値が実行文字集合の同じ文字定数の値と一致するかどうか。（6.10.1）
条件付き取り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。

(57) 条件付き取り込みを制御する定数式中の単一文字から成る文字定数が負の値を持ってもよいかどうか。（6.10.1）
単なる char 型（signed も unsigned もつかない char）が unsigned の場合は，負の値を持ちません。signed の場合は，負の値を持つことがあります。

(58) #include 指令で < と > で囲まれたヘッダが探索される場所，場所の指定方法，及びヘッダの識別方法。（6.10.2）
次の順序で探索し，フォルダにある同名ファイルをヘッダと識別します。
(1) フルパス指定の場合はそのパスが示すフォルダ
(2) オプション -I で指定されたフォルダ
(3) 標準インクルードファイルフォルダ（コンパイラが置かれた bin フォルダからの相対パスでの ..¥inc フォルダ）

(59) #include 指令で，2 つの " で囲まれたソース・ファイルの探索手順。（6.10.2）
次の順序で探索します。
(1) フルパス指定の場合はそのパスが示すフォルダ
(2) ソース・ファイルがあるフォルダ
(3) オプション -I で指定されたフォルダ
(4) 標準インクルードファイルフォルダ（コンパイラが置かれた bin フォルダからの相対パスでの ..¥inc フォルダ）

(60) #include 指令の前処理トークン（マクロ展開の可能性がある）をヘッダ名に結合する方法。（6.10.2）
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，単一のヘッダまたはソース・ファイル名の前処理字句として扱われます。

(61) #include 指令の入れ子の限界。（6.10.2）
制限はありません。

(62) # 演算子が，文字定数や文字列リテラル中の国際文字名の最初の ¥ 文字の前に ¥ 文字を挿入するかどうか。（6.10.3.2）
最初の ¥ 文字の前に ¥ 文字は挿入しません。

(63) 非 STDC のプラグマ指令の動作。（6.10.6）
「4.2.4 #pragma 指令」を参照してください。

(64) 翻訳の日付や時刻が得られない場合，__DATE__ マクロと __TIME__ マクロで得られる日付や時刻。（6.10.8）
日付や時刻が得られない場合はありません。

(65) ISO/IEC 9899:1999 の 4 章で要求される最低限必要なライブラリ機能以外で，フリースタンディング環境でプログラムが利用できるライブラリ機能。（5.1.2.1）
「7. ライブラリ関数仕様」を参照してください。

(66) assert マクロによって標準エラーストリームに書き込まれる診断機能の書式。（7.2.1.1）
以下の通りです。
Assertion failed : *式* , function *関数名* , file *ファイル名* , line *行番号*

(67) fegetexceptflag 関数によって格納される浮動小数点状態フラグの表現。（7.6.2.2）
fegetexceptflag 関数はサポートしていません。

(68) feraiseexcept 関数が，" オーバーフロー" 浮動小数点例外または " アンダーフロー" 浮動小数点例外を生成する場合はいつでも，それに加えて " 不正確結果 " 浮動小数点例外を生成するかどうか。（7.6.2.3）
feraiseexcept 関数はサポートしていません。

(69) setlocale 関数に第 2 引数として渡される文字列で "C" や "" 以外の文字列。（7.11.1.1）
setlocale 関数はサポートしていません。

(70) FLT_EVAL_METHOD マクロの値が 0 未満または 2 より大きい場合の float_t と double_t で定義される型。（7.12）
float_t は float 型，double_t は double 型になります。

(71) 数学関数における，この国際標準（C99）によって要求される以外の定義域エラー。（7.12.1）
下記の場合に定義域エラーとなります。

- cos，sin，tan 関数群の引数が NaN または ± ∞の場合
- atan，fabs，ceil，floor 関数群の引数が NaN の場合
- atan2 関数群の引数のどちらかが NaN，引数の両方が ± ∞の場合
- frexp，modf 関数群の引数 val が NaN または ± ∞の場合
- ldexp 関数群の引数 val が NaN の場合
- scalbn，scalbln 関数群の引数 x が NaN の場合
- fmod 関数群の引数のどちらかが NaN または ± ∞の場合

詳細は (72) を参照してください。

(72) 定義域エラー発生時の数学関数の返却値。（7.12.1）
次の表は，定義域エラーが発生する条件と戻り値をまとめたものです。

関数	定義域エラー発生条件と戻り値
cos/cosf/cosl sin/sinf/sinl tan/tanf/tanl	引数が NaN または ± ∞の場合に NaN を返します。
atan/atanf/atanl fabs/fabsf/fabsl ceil/ceilf/ceill floor/floorf/floorl	引数が NaN の場合に NaN を返します。
atan2/atan2f/atan2l	引数のどちらかが NaN か，引数の両方が ± ∞の場合に NaN を返します。
frexp/frexpf/frexpl	引数 val が NaN または ± ∞の場合に NaN を返し，引数 *exp に 0 を返します。
ldexp/ldexpf/ldexpl	引数 val が NaN の場合に NaN を返します。
scalbn/scalbnf/scalbnl scalbln/scalblnf/scalblnl	引数 x が NaN の場合に NaN を返します。
fmod/fmodf/fmodl	引数 x または引数 y が NaN である場合に NaN を返します。 引数 x が ± ∞，または引数 y が 0 である場合に NaN を返します。 引数 x が ± ∞でなく引数 y が ± ∞である場合に x を返します。
modf/modff/modfl	引数 val が ± ∞の場合に 0 を返し，引数 iptr に ± ∞を返します。 引数 val が NaN の場合に NaN を返し，引数 iptr に NaN を返します。
その他の関数	C99 規格に沿って定義域エラーが発生する場合に NaN を返します。

(73) 浮動小数点演算の結果がアンダーフローした場合の数学関数の返却値。アンダーフロー時，整数式 math_errhandling & MATH_ERRNO が 0 以外の値の場合，整数式 errno の値が ERANGE となるかどうか。アンダーフロー時，整数式 math_errhandling & MATH_ERREXCEPT が 0 以外の値の場合，" アンダーフロー" 浮動小数点例外を生成するかどうか。（7.12.1）
返却値は 0 または非正規化数です。アンダーフロー発生時は errno に ERANGE を設定します。アンダーフロー浮動小数点例外は生成しません。

(74) fmod 関数群の第 2 実引数が 0 の場合に，定義域エラーが発生するか，あるいは 0 が返されるかどうか。（7.12.10.1）
定義域エラーが発生します。詳細は fmod 関数群の説明を参照してください。

(75) remquo 関数群によって商を縮小する際に使用される法の，2 を底とする対数の値。（7.12.10.3）
remquo 関数群はサポートしていません。

(76) シグナル発生時，signal(sig, SIG_DFL); と同等のことが実行されるか，あるいは，その時点のシグナル処理ルーチンが完了するまで，シグナルの処理系定義の組に対してそれらの発生を遮るか。（7.14.1.1）
シグナル操作関数はサポートしていません。

(77) マクロ NULL 展開後の空ポインタ定数。（7.17）
(void*)0 とします。

(78) テキストストリームの最終行が終端を示す改行文字を必要とするかどうか。（7.19.2）
改行文字を必要としません。

(79) テキストストリームにおいて，改行文字の直前に書き込まれた空白文字の並びが，データが読み取られる時に現れるかどうか。（7.19.2）
データが読み取られる時に現れます。

(80) バイナリストリームの最後に付加される NULL 文字の数。（7.19.2）
0 個です。

(81) 追加モードでオープンされたファイルに対し，最初にファイル位置指定子がファイルの始めに位置付けられるか終わりに位置付けられるか。（7.19.3）
ファイル操作関数はサポートしていません。

(82) テキストストリームへの書込みが，結び付けられたファイルを最終書込み点の直後で切り捨てるかどうか。（7.19.3）
ファイル操作関数はサポートしていません。

(83) ファイルバッファリングの特性。（7.19.3）
ファイル操作関数はサポートしていません。

(84) 長さ 0 のファイルが実際に存在するかどうか。（7.19.3）
ファイル操作関数はサポートしていません。

(85) 正しいファイル名の規則。（7.19.3）
ファイル操作関数はサポートしていません。

(86) 同一のファイルを同時に複数回オープンできるかどうか。（7.19.3）
ファイル操作関数はサポートしていません。

(87) ファイル内の多バイト文字のために使用される表現形式の特性と選択。（7.19.3）
ファイル操作関数はサポートしていません。

(88) オープンされているファイルに対する remove 関数の効果。（7.19.4.1）
ファイル操作関数はサポートしていません。

(89) rename 関数を呼び出す前に，第 2 引数で指定された新しいファイル名のファイルが既に存在していた場合の rename 関数の動作。（7.19.4.2）
ファイル操作関数はサポートしていません。

(90) プログラムが異常終了した場合，オープン中の一時ファイルが削除されるかどうか。（7.19.4.3）
ファイル操作関数はサポートしていません。

(91) filename 引数が空ポインタの場合，どのようなモード変更を許すか，及びどのような状況での変更を許すか。（7.19.5.4）
ファイル操作関数はサポートしていません。

(92) 無限大や NaN を書き込む時の形式，及び NaN の書込みで使われるかもしれない n 文字列，n ワイド文字列の意味。（7.19.6.1，7.24.2.1）
正の無限大は inf または INF，負の無限大は -inf または -INF，NaN は nan または NAN を出力します。
NaN の書き込みにおける n 文字列，n ワイド文字列はサポートしていません。

(93) fprintf 関数および fwprintf 関数での %p 変換の出力。（7.19.6.1，7.24.2.1）
16 進表記です。
fprintf 関数，fwprintf 関数はサポートしていません。

(94) fscanf 関数および fwscanf 関数の %[変換において，文字 - が走査文字の並びに含まれ，かつ先頭の文字（先頭が文字 ^ のときは 2 文字目）でも最後の文字でもない場合の文字 - の解釈。（7.19.6.2，7.24.2.1）
「7.5.7 標準入出力関数」にある「scanf」を参照してください。
fscanf 関数，fwscanf 関数はサポートしていません。

(95) fscanf 関数および fwscanf 関数での %p 変換によって一致する文字の並びの集合と対応する入力項目の解釈。（7.19.6.2，7.24.2.2）
16 進数です。
fscanf 関数，fwscanf 関数はサポートしていません。

(96) fgetpos 関数，fsetpos 関数，ftell 関数の失敗時に設定される errno マクロの値。（7.19.9.1，7.19.9.3，7.19.9.4）
ファイル操作関数はサポートしていません。

(97) strtod 関数，strtof 関数，strtold 関数，wcstod 関数，wcstof 関数，wcstold 関数によって変換された NaN を表現する文字列での n 文字あるいは n ワイド文字の並びの意味。（7.20.1.3，7.24.4.1.1）
strtod 関数，strtof 関数，strtold 関数は浮動小数点型の数値以外と解釈します。
wcstod 関数，wcstof 関数，wcstold 関数はサポートしていません。

(98) strtod 関数，strtof 関数，strtold 関数，wcstod 関数，wcstof 関数，wcstold 関数が，アンダーフロー発生時に errno に ERANGE 格納するかどうか。（7.20.1.3，7.24.4.1.1）
strtod 関数，strtof 関数，strtold 関数はグローバル変数 errno に ERANGE を設定します。
wcstod 関数，wcstof 関数，wcstold 関数はサポートしていません。

(99) calloc 関数，malloc 関数，realloc 関数が，要求された領域の大きさが 0 である時に，割り付けたオブジェクトへのポインタを返すか，それとも空ポインタを返すか。（7.20.3）
NULL を返します。

(100) abort 関数，_Exit 関数の呼び出し時に，書き出されていないバッファリングされたデータをもつオープンしているストリームをフラッシュするかどうか，オープンしているストリームをクローズするかどうか，一時ファイルを削除するかどうか。（7.20.4.1，7.20.4.4）
ファイル操作関数はサポートしていません。

(101) abort 関数，exit 関数，_Exit 関数によってホスト環境へ返される終了状態。（7.20.4.1，7.20.4.3，7.20.4.4）
フリースタンディング環境であるため，規定しません。

(102) system 関数に渡された実引数が空ポインタでない場合に，system 関数によって返される値。（7.20.4.6）
system 関数はサポートしていません。

(103) 地方時と夏時間。（7.23.1）
time.h はサポートしていません。

(104) clock_t と time_t で表現可能な時刻の範囲と精度。（7.23）
time.h はサポートしていません。

(105) clock 関数で計算されるプロセッサ時間の開始時点。（7.23.2.1）
time.h はサポートしていません。

(106) "C" ロケールでの，strftime 関数，wcsftime 関数の %Z 変換指定子に対応する置換文字列。（7.23.3.5，7.24.5.1）
time.h はサポートしていません。

(107) IEC 60559 準拠の処理系で，三角関数，双曲線関数，底が e の指数関数，底が e の対数関数，エラー関数，ログガンマ関数が，" 不正確結果 " 浮動小数点例外を生成するかどうか。生成する場合，いつ生成するか。（F.9）
" 不正確結果 " 浮動小数点例外を生成しません。

(108) IEC 60559 準拠の処理系で，<math.h> の関数が丸め方向モードを尊重するか。（F.9）
丸め方向モードは固定です。
fesetround 関数はサポートしていません。

(109) <float.h>，<limits.h>，<stdint.h> の各ヘッダで指定されたマクロに割り当てられる値あるいは式。（5.2.4.2，7.18.2，7.18.3）
「4.2.3 C90 でサポートする C99 言語仕様」にある「標準ヘッダ」を参照してください。

(110) 明示的に規定されていない場合の，オブジェクトを構成するバイトの並びのバイト数，バイトの順序，表現方法。（6.2.6.1）
「4.1.3 データの内部表現と領域」を参照してください。

(111) sizeof 演算子の結果の値。（6.5.3.4）
「4.1.3 データの内部表現と領域」を参照してください。

翻訳限界

CC-RL の翻訳限界を以下に示します。
「制限なし」の項目は，上限はホスト環境のメモリ状況に依存します。

表 4.2 翻訳限界（C99）

項目	C99	CC-RL
ブロックの入れ子のレベル数	127	制限なし
条件付取り込みにおける入れ子のレベル数	63	制限なし
一つの宣言中の一つの算術型，構造体型，共用体型又は不完全型を修飾するポインタ，配列及び関数宣言子（の任意の組み合わせ）の個数	12	128
一つの完全宣言子における括弧で囲まれた宣言子の入れ子のレベル数	63	制限なし
一つの完全式における括弧で囲まれた式の入れ子のレベル数	63	制限なし
内部識別子又はマクロ名において意味がある先頭の文字数	63	制限なし
外部識別子において意味がある先頭の文字数	31	制限なし
一つの翻訳単位中における外部識別子数	4095	制限なし
一つのブロック中におけるブロック有効範囲をもつ識別子数	511	制限なし
一つの翻訳単位中で同時に定義されうるマクロ識別子数	4095	制限なし
一つの関数定義における仮引数の個数	127	制限なし
一つの関数呼出しにおける実引数の個数	127	制限なし
一つのマクロ定義における仮引数の個数	127	制限なし
一つのマクロ呼出しにおける実引数の個数	127	制限なし
一つの論理ソース行における文字数	4095	制限なし
（連結後の）単純文字列リテラル又はワイド文字列リテラル中のおける文字数	4095	制限なし
（ホスト環境の場合）一つのオブジェクトのバイト数	65535	32767（65535）注 1
#include で取り込まれるファイルの入れ子レベル数	15	制限なし
一つの switch 文（入れ子になった switch 文を除く）中における case 名札の個数	1023	65535
一つの構造体又は共用体のメンバ数	1023	制限なし
一つの列挙体における列挙定数の個数	1023	制限なし
一つのメンバ宣言並びにおける構造体又は共用体定義の入れ子のレベル数	63	制限なし

注 1. カッコ内は -large_variable 指定時

4.1.3 データの内部表現と領域

この項では，CC-RL 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。

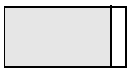
(1) 整数型

(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。

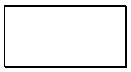
図 4.1 整数型の内部表現

_Bool



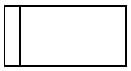
ゼロビット目のみが意味を持ちます。1 ～ 7 ビット目は不定値となります。
オプション -lang=c かつ -strict_std 使用時は，_Bool 型は C90 違反でエラーとなります。

char

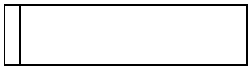


signed も unsigned もつかない単なる char 型は，unsigned char と同じ表現とします。
ただし，オプション -signed_char 使用時は，signed char と同じ表現とします。

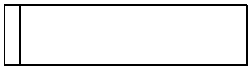
signed char（unsigned では符号ビットなし）



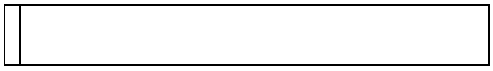
short（unsigned では符号ビットなし）



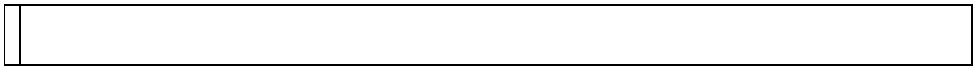
int（unsigned では符号ビットなし）



long（unsigned では符号ビットなし）



long long（unsigned では符号ビットなし）



オプション -lang=c かつ -strict_std 使用時は，long long 型は C90 違反でエラーとなります。

(b) 値域

表 4.3 整数型の値域

型	値域
_Bool	0 ～ 1
signed char	-128 ～ +127
signed short	-32768 ～ +32767
signed int	-32768 ～ +32767
signed long	-2147483648 ～ +2147483647
signed long long	-9223372036854775808 ～ +9223372036854775807
(unsigned) char	0 ～ 255
unsigned short	0 ～ 65535
unsigned int	0 ～ 65535
unsigned long	0 ～ 4294967295
unsigned long long	0 ～ 18446744073709551615

(c) 整数定数

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 4.4 整数定数の型（long long 型が有効な場合（-lang=c 指定あり，-strict_std 指定なし））

接尾語	10 進定数	2 進定数，8 進定数，または 16 進定数
なし	int long int unsigned long int 注 long long int unsigned 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 unsigned long int 注 long long int unsigned 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 unsigned 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。これは C90 であれば 4 バイト・データであるところを，8 バイト・データになることを回避するためです。

表 4.5 整数定数の型（long long 型が無効な場合（-lang=c 指定あり，-strict_std 指定あり））

接尾語	10 進定数	2 進定数，8 進定数，または 16 進定数
なし	int long int unsigned long int	int unsigned int long int unsigned long int
u，または U	unsigned int unsigned long int	unsigned int unsigned long int
l，または L	long int unsigned 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

表 4.6 整数定数の型（long long 型が有効な場合（-lang=c99 指定あり））

接尾語	10 進定数	2 進定数，8 進定数，または 16 進定数
なし	int long int long long int unsigned long long int	int unsigned int long int unsigned long int long long int unsigned long long int
u，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 unsigned 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 unsigned 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

(2) 浮動小数点型

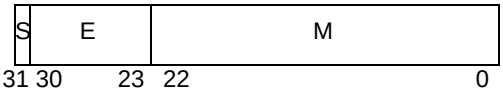
(a) 内部表現

浮動小数点型データの内部表現は，IEC 60559:1989 (IEEE 754-1985) [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。

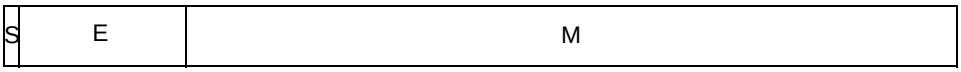
図 4.2 浮動小数点型の内部表現

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）

オプション -dbl_size=4 使用時は float 型と同じ表現とします。double 型を書いても float 型を書いたものとして扱います。long double 型も同様に float 型を書いたものとして扱います。
オプション -dbl_size=8 使用時は，64 ビットで表現されます。

(b) 値域

表 4.7 浮動小数点型の値域

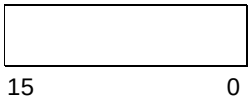
型	値域
float	1.17549435E-38F ～ 3.40282347E+38F
double	2.2250738585072014E-308 ～ 1.7976931348623158E+308
long double	2.2250738585072014E-308 ～ 1.7976931348623158E+308

(3) ポインタ型

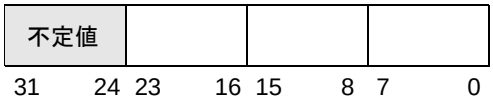
(a) 内部表現
near ポインタの内部表現は 16 ビットの符号なし型，far ポインタの内部表現は 32 ビットの符号なし型となります。
far ポインタの最上位 1 バイトは不定値となります。
空ポインタ定数（NULL）の内部表現は値 0 となります（ただし，far ポインタの不定値部分は 0 とは限りません）。
このため，near ポインタ，far ポインタ共にゼロ番地に対するアクセスは保証しません。
far ポインタの値が 0xfffff を越えた場合の動作は保証しません。
0x0f0000 番地に関数や変数を配置したり，アクセスすることはしないでください。

図 4.3 ポインタ型の内部表現

near ポインタ



far ポインタ



(4) 列挙型

(a) 内部表現
列挙型の内部表現は，列挙子の値の範囲によって変化します。

<1> オプション -signed_char なし

列挙子の下限値	列挙子の上限値	内部表現の型	備考
-128	127	signed char	
0	255	char	0 ～ 127 の場合はここに含む

列挙子の下限値	列挙子の上限値	内部表現の型	備考
上記以外		signed short	

<2> オプション -signed_char あり

列挙子の下限値	列挙子の上限値	内部表現の型	備考
-128	127	char	0 ～ 127 の場合はここに含む
0	255	unsigned char	
上記以外		signed short	

(5) 配列型

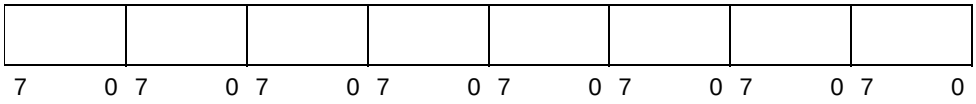
(a) 内部表現

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 4.4 配列型の内部表現



(6) 構造体型

(a) 内部表現

ひとつの構造体において , 各メンバは構造体先頭から宣言順に配置されます。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。
構造体全体の整列条件は，その構造体で最も大きなメンバの整列条件に合わせたものとなります。このルールは，メンバが構造体や共用体の場合にも再帰的に適用されます。
構造体のサイズは「構造体全体の整列条件」の倍数となります。したがって，構造体の末尾が構造体自身の整列条件とあわない場合に次の整列条件を保証するために作成される未使用領域もサイズに含まれます。

例 1.

```
struct {
        short           s1;
        signed long     s2;
        char            s3;
        signed long     s4;
} s;
```

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 4.5 構造体型の内部表現（構造体パッキング指定なし）

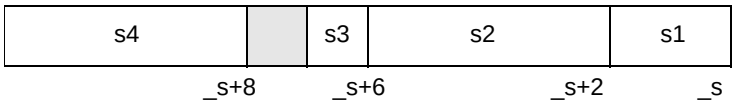
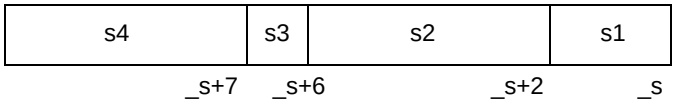


図 4.6 構造体型の内部表現（構造体パッキング指定あり）



例 2.

```
struct {
        short           s1;
        char            s2;
} s;
```

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 4.7 構造体型の内部表現（構造体パッキング指定なし）

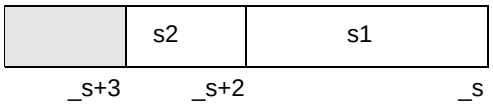
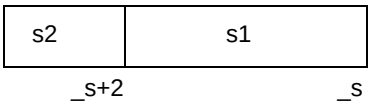


図 4.8 構造体型の内部表現（構造体パッキング指定あり）



構造体パッキング指定については，「-pack」を参照してください。

(7) 共用体型

(a) 内部表現

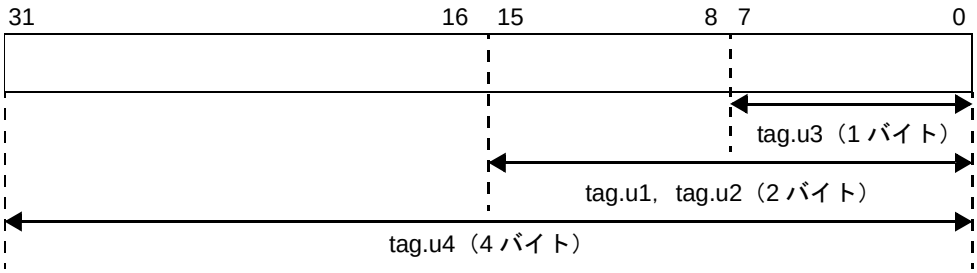
共用体全体の整列条件は，その共用体で最も大きなメンバの整列条件に合わせたものとなります。このルールは，メンバが構造体や共用体の場合にも再帰的に適用されます。

例

```
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 4.9 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・フィールドには _Bool，char，signed char，unsigned char，signed short，unsigned short，signed int，unsigned int，signed long，unsigned long，signed long long，unsigned long long，および列挙型を指定することができます。

C90 では（signed ／ unsigned）int 型のみ許されていますが，CC-RL ではオプション -strict_std を使用しない場合，上記の型を全てビット・フィールドに対して有効とします。オプション -strict_std を使用する場合，_Bool，signed long long，unsigned longl long 型は C90 違反でエラーとします。

ビット・フィールドは，宣言した型の最下位ビットから割り当てます。

ビット・フィールドを直前のビット・フィールドに続くビットから配置すると，配置後のビット・フィールドの末尾の位置が，ビット・フィールドの整列条件を満たす直前の境界に「宣言した型のビット幅」を加えた位

置を超える場合，そのビット・フィールドは直前のビット・フィールド以降にあるビット・フィールドの整列条件を満たす最初の境界に整列します。

- signed も unsigned もつかない型のビット・フィールドは，符号なしとして扱われます。
  ただし，オプション -signed_bitfield 使用時は，符号付きとして扱われます。
- 割り付け単位内のビット・フィールドの割り付け順序は下位（LSB）から上位（MSB）となります。

例 1.

```
struct S{
        char            a;
        char            b:2;
        signed char     c:3;
        unsigned char   d:4;
        int             e;
        short           f:5;
        int             g:6;
        unsigned char   h:2;
        unsigned int    i:2;
};
```

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 4.10 ビット・フィールドの内部表現（構造体パッキング指定なし）

sizeof(struct S)=8

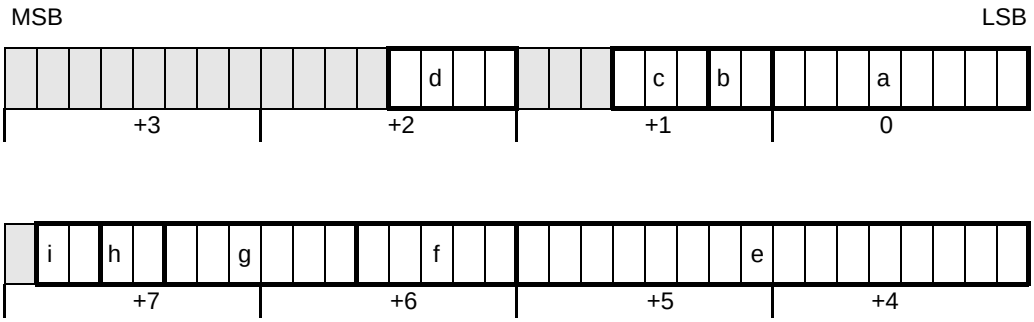
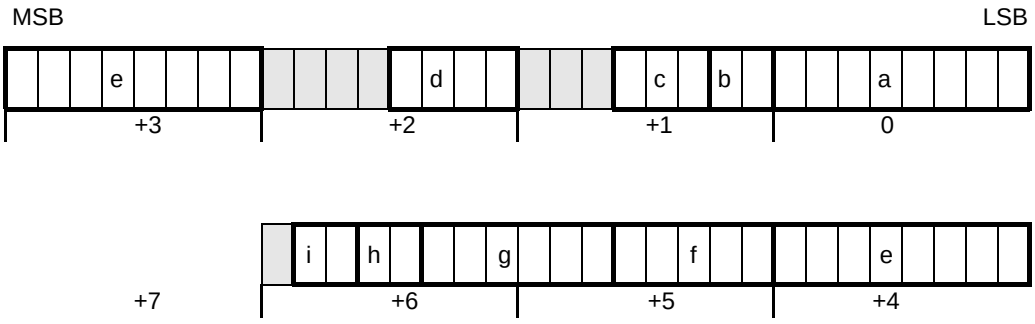


図 4.11 ビット・フィールドの内部表現（構造体パッキング指定あり）

sizeof(struct S)=7



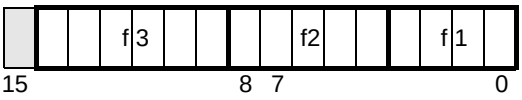
例 2.

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

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

sizeof(struct S)=2



例 3.

```
struct S{
        long    f1:4;
};
```

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（構造体パッキング指定なし）

sizeof(struct S)=2

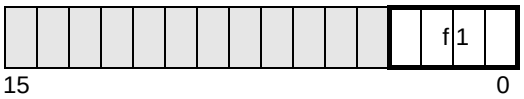
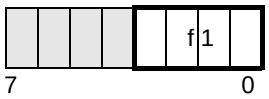


図 4.14 ビット・フィールドの内部表現（構造体パッキング指定あり）

sizeof(struct S)=1



構造体パッキング指定については，「-pack」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。

表 4.8 基本型に対する整列条件

基本型	整列条件
（unsigned）char _Bool 型	1 バイト境界
上記以外	2 バイト境界

(b) 列挙型に対する整列条件
次に，列挙型に対する整列条件を示します。

<1> オプション -signed_char なし

列挙子の下限値	列挙子の上限値	内部表現の型	整列条件
-128	127	signed char	1
0	255	char	1
上記以外		signed short	2

<2> オプション -signed_char あり

列挙子の下限値	列挙子の上限値	内部表現の型	整列条件
-128	127	char	1
0	255	unsigned char	1
上記以外		signed short	2

(c) 配列に対する整列条件
配列型に対する整列条件は，配列要素の整列条件と同じになります。

(d) ポインタに対する整列条件
near ポインタ，far ポインタに対する整列条件は，2 となります。

(e) 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，共用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。

(f) 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。

(g) 関数引数に対する整列条件
「9.1 関数呼び出しインタフェース」を参照してください。

(h) 関数に対する整列条件
関数に対する整列条件は，1 バイト境界となります。

4.1.4 データ，および関数の配置条件

データ，および関数の配置条件は次のようになります。

- 1 つのセクション内における静的変数や関数の配置順序は，それらの定義順となります。
すなわち，あるデータ（または関数）A とデータ（または関数）B が同じセクション S 内に配置され，プログラム内で A の定義が B の定義よりも前にある場合，セクション S 内における A のアドレスは B のアドレスよりも小さい値となります。
- 静的変数や関数の整列は，セクションの種類によらず静的変数や関数の整列条件に従います。
すなわち，整列条件が 1 の変数は，整列条件が 2 のセクションに配置されても，1 バイト境界に配置され，整列条件が 2 の変数は，整列条件が 1 のセクションに配置できません。

4.1.5 静的変数の初期化

この項では，静的変数の初期化について説明します。

4.1.5.1 アドレス演算による初期化

静的変数を以下の方法で初期化した場合，加減算の効果がおよぶのは下位 2 バイトのみとなります。

((整数型キャスト) アドレス定数)± 整数型定数

この指定方法は -strict_std が指定されない場合だけ有効となります。

例

```
int     x;
static long     l = (long)&x + 1;               //&x の上位 2 バイトは変化しない
```

4.1.5.2 far アドレスを near アドレスにキャスト後，さらに far アドレスに変換する場合

静的変数を以下の方法で初期化した場合，near ポインタへのキャストによる上位 2 バイトの欠落は発生しません。

(4 バイト以上へのキャスト)(near ポインタへのキャスト)(far アドレス定数)

この指定方法は -strict_std が指定されない場合だけ有効となります。

例

```
int     __far x;
static long     l = (long)(int __near*)&x;      //l には far アドレスが入る
```

4.2 拡張言語仕様

この節では，CC-RL 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-RL 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-RL で追加されている予約語一覧を示します。

表 4.9 予約語一覧

予約語	用途
__saddr	saddr 領域に静的変数を割り当てます。
__callt	callt 命令で関数を呼び出します。
__near	メモリ配置領域指定
__far	メモリ配置領域指定
__inline	インライン関数指定
__sectop	セクションの先頭アドレス
__secend	セクションの末尾アドレス +1

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 マクロ

以下に，サポートしているマクロ名を示します。

表 4.10 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）。注 2
__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）。注 2
__STDC__	10 進定数 1（-strict_std オプションを指定した場合に定義）。注 1
__RENESAS__	10 進定数 1。
__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
__RL78__	10 進定数 1。
__RL78_S1__	10 進定数 1（-cpu オプションで S1 を指定した場合に定義）。
__RL78_S2__	10 進定数 1（-cpu オプションで S2 を指定した場合に定義）。
__RL78_S3__	10 進定数 1（-cpu オプションで S3 を指定した場合に定義）。

マクロ名	定義
__RL78_SMALL__	10 進定数 1（-memory_model オプションで small を指定した場合，あるいは，-memory_model オプションを指定せず，-cpu オプションで S1 を指定した場合に定義）。
__RL78_MEDIUM__	10 進定数 1（-memory_model オプションで medium を指定した場合，あるいは，-memory_model オプションを指定せず，-cpu オプションで S2，または S3 を指定した場合に定義）。
__CCRL__	10 進定数 1。
__CCRL	10 進定数 1。
__DBL4	10 進定数 1（-dbl_size オプションで 4 を指定した場合に定義）。
__DBL8	10 進定数 1（-dbl_size オプションで 8 を指定した場合に定義）。
__SCHAR	10 進定数 1（-signed_char オプションを指定した場合に定義）。
__UCHAR	10 進定数 1（-signed_char オプションを指定しなかった場合に定義）。
__SBIT	10 進定数 1（-signed_bitfield オプションを指定した場合に定義）。
__UBIT	10 進定数 1（-signed_bitfield オプションを指定しなかった場合に定義）。
__FAR_ROM__	10 進定数 1（-far_rom オプションを指定した場合に定義）。
__CNV_CA78K0R__	10 進定数 1（-convert_cc オプションで ca78k0r を指定した場合に定義）。
__CNV_NC30__	10 進定数 1（-convert_cc オプションで nc30 を指定した場合に定義）。
__CNV_IAR__	10 進定数 1（-convert_cc オプションで iar を指定した場合に定義）。
__BASE_FILE__	C ソース・ファイルの名前（文字列定数）。 __FILE__ と異なり，インクルード・ファイル内で使用しても C ソース・ファイル名を返します。
__STDC_VERSION__	10 進定数 199409L（-lang=c かつ -strict_std オプションを指定した場合に定義）。注 1 10 進定数 199901L（-lang=c99 オプションを指定した場合に定義）。
__STDC_HOSTED__	10 進定数 0（-lang=c99 オプションを指定した場合に定義）。
__STDC_IEC_559__	10 進定数 1（-lang=c99 オプションを指定した場合に定義）。

注 1. -strict_std オプション指定時の処理については，「 -strict_std【V1.06 以降】/-ansi【V1.05 以前】」を参照してください。

注 2. 翻訳日付または翻訳時刻を得られない場合はなく，__DATE__，__TIME__ は常に定義を持ちます。

4.2.3 C90 でサポートする C99 言語仕様

CC-RL では，C90 準拠時（-lang=c 指定時）でも，一部の C99 規格の仕様が有効です。

(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。
_Bool 型は 1 バイトの整数型ですが，0 または 1 のみを保持します。
オプション -lang=c かつ -strict_std 使用時は _Bool 型をサポートしないので，コンパイル・エラーとなります。

(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 および ULL 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。
オプション -lang=c かつ -strict_std 使用時は long long int 型をサポートしないので，コンパイル・エラーとなります。

(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。
オプション -lang=c かつ -strict_std 使用時は _Bool 型と long long 型をサポートしないので，C90 の仕様に従います。

(6) 集成体の初期化
自動記憶域期間を持つ集成体型オブジェクトおよび共用体型オブジェクトの初期化子は，C99 の 仕様に従います。
C90 仕様では，初期化子に定数式のみを使用可能ですが，定数式以外も使用可能とします。

```
void func(int param) {
        int i = param;              //C90，C99 ともに OK
        int array[] = { param };    //C90 では NG，C99 では OK，CC-RL ではともに OK
```

(7) 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（2 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。
- オプション -dbl_size=4 使用時は，float 型は 4 バイトのまま関数を呼び出します。
  これはオプションの作用により，float 型を double 型に拡張しても，double 型が 4 バイト（float 型に等しい）であるためです。

near ポインタは，far ポインタに変換されます。
void* は変数ポインタの規則に従います。

(8) 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,};
```

オプション -lang=c かつ -strict_std 使用時はエラーとなります。

(9) 整数定数の型
long long 型の追加に伴い，整数定数の型が変わります。詳細は「4.1.3 データの内部表現と領域」の「(c) 整数定数」を参照してください。

(10) 標準ヘッダ

(a) limits.h

表 4.11 limits.h

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	0（-128）	char 型の最小値（デフォルトは unsigned。オプション -signed_char で signed になります。）
CHAR_MAX	+255（+127）	char 型の最大値（デフォルトは unsigned。オプション -signed_char で signed になります。）
SHRT_MIN	-32768	short int 型の最小値
SHRT_MAX	+32767	short int 型の最大値

名前	値	意味
USHRT_MAX	+65535	unsigned short int 型の最大値
INT_MIN	-32768	int 型の最小値
INT_MAX	+32767	int 型の最大値
UINT_MAX	+65535	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 （オプション -lang=c かつ -strict_std 使用時は無効）
LLONG_MAX	+9223372036854775807	long long int 型の最大値 （オプション -lang=c かつ -strict_std 使用時は無効）
ULLONG_MAX	+18446744073709551615	unsigned long long int 型の最大値 （オプション -lang=c かつ -strict_std 使用時は無効）

(b) float.h

（）内は，sizeof(double)=sizeof(long double)=4 にするオプション（-dbl_size=4）指定時の値です。CC-RL は，-dbl_size=4 をデフォルト動作とします。

表 4.12 float.h

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（もっとも近傍へ丸める）とします。
FLT_EVAL_METHOD	0	浮動小数点数の評価形式 （-lang=c 指定時は無効）
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 を底とする数字の桁数（p）
DBL_MANT_DIG	+53（+24）	
LDBL_MANT_DIG	+53（+24）	
DECIMAL_DIG	+17（+9）	基数 b の p 桁をもつ浮動小数点数を q 桁の 10 進数の浮動小数点数に丸めることができ，再び変更なしに基数 b の p 桁をもつ浮動小数点数に戻すことが可能な 10 進数の桁数（q） （-lang=c 指定時は無効）
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（+6）	
LDBL_DIG	+15（+6）	
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（-125）	
LDBL_MIN_EXP	-1021（-125）	

名前	値	意味
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（-37）	
LDBL_MIN_10_EXP	-307（-37）	
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（+128）	
LDBL_MAX_EXP	+1024（+128）	
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（+38）	
LDBL_MAX_10_EXP	+308（+38）	
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 （3.40282347E+38F）	
LDBL_MAX	1.7976931348623158E+308 （3.40282347E+38F）	
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 （1.19209290E - 07F）	
LDBL_EPSILON	2.2204460492503131E-016 （1.19209290E - 07F）	
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 （1.17549435E - 38F）	
LDBL_MIN	2.2250738585072014E-308 （1.17549435E - 38F）	

(c) stdint.h

表 4.13 stdint.h の型定義名

型名	実際の型	備考
int8_t	signed char	
int16_t	signed short	
int32_t	signed long	
int64_t	signed long long	-strict_std 不使用時， または -lang=c99 使用時
uint8_t	unsigned char	
uint16_t	unsigned short	
uint32_t	unsigned long	
uint64_t	unsigned long long	-strict_std 不使用時， または -lang=c99 使用時
int_least8_t	signed char	

型名	実際の型	備考
int_least16_t	signed short	
int_least32_t	signed long	
int_least64_t	signed long long	-strict_std 不使用時, または -lang=c99 使用時
uint_least8_t	unsigned char	
uint_least16_t	unsigned short	
uint_least32_t	unsigned long	
uint_least64_t	unsigned long long	-strict_std 不使用時, または -lang=c99 使用時
int_fast8_t	signed char	
int_fast16_t	signed short	
int_fast32_t	signed long	
int_fast64_t	signed long long	-strict_std 不使用時, または -lang=c99 使用時
uint_fast8_t	unsigned char	
uint_fast16_t	unsigned short	
uint_fast32_t	unsigned long	
uint_fast64_t	unsigned long long	-strict_std 不使用時, または -lang=c99 使用時
intptr_t	signed long	
uintptr_t	unsigned long	
intmax_t	signed long	-lang=c 使用, かつ -strict_std 使用時
	signed long long	-strict_std 不使用時, または -lang=c99 使用時
uintmax_t	unsigned long	-lang=c 使用, かつ -strict_std 使用時
	unsigned long long	-strict_std 不使用時, または -lang=c99 使用時

表 4.14 stdint.h のマクロ定義名

マクロ名	値	備考
INT8_MIN	-128	
INT16_MIN	-32768	
INT32_MIN	-2147483648	
INT64_MIN	-9223372036854775808	-strict_std 不使用時, または -lang=c99 使用時
INT8_MAX	+127	
INT16_MAX	+32767	

マクロ名	値	備考
INT32_MAX	+2147483647	
INT64_MAX	+9223372036854775807	-strict_std 不使用時, または -lang=c99 使用時
UINT8_MAX	+255	
UINT16_MAX	+65535	
UINT32_MAX	+4294967295	
UINT64_MAX	+18446744073709551615	-strict_std 不使用時, または -lang=c99 使用時
INT_LEAST8_MIN	-128	
INT_LEAST16_MIN	-32768	
INT_LEAST32_MIN	-2147483648	
INT_LEAST64_MIN	-9223372036854775808	-strict_std 不使用時, または -lang=c99 使用時
INT_LEAST8_MAX	+127	
INT_LEAST16_MAX	+32767	
INT_LEAST32_MAX	+2147483647	
INT_LEAST64_MAX	+9223372036854775807	-strict_std 不使用時, または -lang=c99 使用時
UINT_LEAST8_MAX	+255	
UINT_LEAST16_MAX	+65535	
UINT_LEAST32_MAX	+4294967295	
UINT_LEAST64_MAX	+18446744073709551615	-strict_std 不使用時, または -lang=c99 使用時
INT_FAST8_MIN	-128	
INT_FAST16_MIN	-32768	
INT_FAST32_MIN	-2147483648	
INT_FAST64_MIN	-9223372036854775808	-strict_std 不使用時, または -lang=c99 使用時
INT_FAST8_MAX	+127	
INT_FAST16_MAX	+32767	
INT_FAST32_MAX	+2147483647	
INT_FAST64_MAX	+9223372036854775807	-strict_std 不使用時, または -lang=c99 使用時
UINT_FAST8_MAX	+255	
UINT_FAST16_MAX	+65535	
UINT_FAST32_MAX	+4294967295	
UINT_FAST64_MAX	+18446744073709551615	-strict_std 不使用時, または -lang=c99 使用時
INTPTR_MIN	-2147483648	

マクロ名	値	備考
INTPTR_MAX	+2147483647	
UINTPTR_MAX	+4294967295	
INTMAX_MIN	-2147483648	-lang=c 使用, かつ -strict_std 使用時
	-9223372036854775808	-strict_std 不使用時, または -lang=c99 使用時
INTMAX_MAX	+2147483647	-lang=c 使用, かつ -strict_std 使用時
	+9223372036854775807	-strict_std 不使用時, または -lang=c99 使用時
UINTMAX_MAX	+4294967295	-lang=c 使用, かつ -strict_std 使用時
	+18446744073709551615	-strict_std 不使用時, または -lang=c99 使用時
PTRDIFF_MIN	-32768	
PTRDIFF_MAX	+32767	
SIZE_MAX	+65535	

(11) 可変個引数マクロ
可変個引数マクロを許可します。

```
#define pf(form,...) printf(form, __VA_ARGS__)

pf("%s %d¥n", "string", 100);
              ↓
printf("%s %d¥n", "string", 100);
```

4.2.4 #pragma 指令

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。C99 言語における _Pragma 演算子でもこれらの拡張機能を使用できます。詳細は「4.2.6 拡張言語仕様の使用方法」を参照してください。

表 4.15 サポートしている #pragma 指令

#pragma 指令	内容
#pragma interrupt	ハードウェア割り込みハンドラ
#pragma interrupt_brk	ソフトウェア割り込みハンドラ
#pragma section	コンパイラ出力セクション名の変更
#pragma rtos_interrupt	RTOS 用割り込みハンドラ
#pragma rtos_task	RTOS 用タスク関数
#pragma inline	関数のインライン展開
#pragma noinline	
#pragma inline_asm	アセンブラ命令の記述
#pragma address	絶対番地配置指定
#pragma saddr	saddr 領域利用

#pragma 指令	内容
#pragma callt	callt 関数
#pragma near	near 関数
#pragma far	far 関数
#pragma pack	構造体のパッキング
#pragma unpack	
#pragma stack_protector	スタック破壊検出コードの生成 【Professional 版のみ】
#pragma no_stack_protector	

4.2.5 2 進定数

CC-RL では，2 進定数をサポートしています。

4.2.6 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。

- saddr 領域利用（__saddr）
- callt 関数（__callt）
- メモリ配置領域指定（__near/__far）
- インライン関数指定（__inline）
- セクション・アドレス演算子（__sectop/__secend）
- ハードウェア割り込みハンドラ（#pragma interrupt）
- ソフトウェア割り込みハンドラ（#pragma interrupt_brk）
- コンパイラ出力セクション名の変更（#pragma section）
- RTOS 用割り込みハンドラ（#pragma rtos_interrupt）
- RTOS 用タスク関数（#pragma rtos_task）
- 関数のインライン展開（#pragma inline/#pragma noinline）
- アセンブラ命令の記述（#pragma inline_asm）
- 絶対番地配置指定（#pragma address）
- saddr 領域利用（#pragma saddr）
- callt 関数（#pragma callt）
- near/far 関数（#pragma near/#pragma far）【V1.05 以降】
- 構造体のパッキング（#pragma pack/#pragma unpack）【V1.05 以降】
- スタック破壊検出コードの生成（#pragma stack_protector/#pragma no_stack_protector）【Professional 版のみ】【V1.02 以降】
- 2 進定数

saddr 領域利用（__saddr）

__saddr 宣言された外部変数，および関数内 static 変数を saddr 領域に割り当てます。

［機能］

- 初期値あり変数はセクション .sdata に配置します。
- 初期値なし変数はセクション .sbss に配置します。
- アドレス参照は，必ず near ポインタを返します。
- __saddr 宣言された外部変数，および関数内 static 変数は，saddr 領域に割り当てます。
- C ソース中において，通常の変数と同様に扱います。

［効果］

- saddr 領域に対する命令は，通常メモリに対する命令よりも短く，オブジェクト・コードが小さくなり，実行速度が向上します。

［方法］

- 変数を定義するモジュール中，および関数の中で，__saddr 宣言を行ないます。
  関数の中では，extern，または static 記憶域クラス指定子が付いている変数のみ宣言することができます。

```
__saddr 変数宣言;
extern __saddr  変数宣言;
static __saddr  変数宣言;
__saddr extern  変数宣言;
__saddr static  変数宣言;
```

［制限］

- const 型変数に __saddr を指定した場合は，コンパイル・エラーとなります。
- 関数に __saddr を指定した場合は，コンパイル・エラーとなります。
- __saddr を静的記憶域期間を持つ変数以外に使用した場合は，コンパイル・エラーとなります。
- __saddr と __near/__far を同時指定した場合は，コンパイル・エラーとなります。

```
__saddr __near int      ni;     // エラー
__saddr __far int       fi;     // エラー
```

- typedef 中に __saddr を使用した場合は，コンパイル・エラーとなります。

```
typedef __saddr int     SI;     // エラー
```

［使用例］

C ソースを以下に示します。

```
__saddr int     hsmm0;          //.sbss
__saddr int     hsmm1=1;        //.sdata
__saddr int     *hsptr;         //hsptr は .sbss に配置する。指す先は通常の領域。
void main(){
        hsmm0 -= hsmm1;
        hsptr = 0;
}
```

アセンブリ・ソースにおける宣言とセクション割り付けは以下となります。

```
        .PUBLIC _hsmm0                  ;宣言
        .PUBLIC _hsmm1                  ;宣言
        .PUBLIC _hsptr                  ;宣言

        .SECTION        .sbss,SBSS      ;.sbss セクションへの割り付け
        .ALIGN  2
_hsmm0:
        .DS     2
        .ALIGN  2
_hsptr:
        .DS     2

        .SECTION        .sdata,SDATA    ;.sdata セクションへの割り付け
        .ALIGN  2
_hsmm1:
        .DB2    0x0001
```

関数中のコードは以下となります。

```
        movw    ax, _hsmm0
        subw    ax, _hsmm1
        movw    _hsmm0, ax
        movw    _hsptr, #0x0000
```

［備考］

- キーワード __saddr と #pragma saddr の違いについて
  - __saddr キーワードは，__near/__far キーワードとの混在を許さずコンパイル・エラーとなります。
  - #pragma saddr は，__near/__far キーワードを付加した変数でも警告なしで __saddr を指定したものとして扱います。

callt 関数（__callt）

__callt 宣言された関数を callt 命令で呼び出します。

［機能］

- __callt 宣言された関数（callt 関数）は，callt 命令で呼び出します。
  callt 命令は，callt 命令テーブルと呼ばれる領域（0x80 ～ 0xBF）に「関数定義の先頭アドレス」が格納された関数を，直接関数を呼ぶよりも短いコードで呼ぶことを可能にします。
- callt 関数の呼び出しには，関数名の先頭に ”@_” を付加したラベル名を使用し，callt 命令で呼び出します。
  関数末尾で callt 関数を呼び出す場合，callt 命令での呼び出しにならない場合があります。
- 呼ばれる関数は，C ソース中において，通常の関数と同様に扱います。
- __near 指定となり，アドレス参照は必ず near ポインタを返します。
- callt 関数は再配置属性 TEXT のセクションへ配置します。
- callt 命令テーブルはセクション .callt0 に配置します。

［効果］

- 2 バイト・コール命令での関数呼び出しとなるため，オブジェクト・コードのサイズが小さくなります。

［方法］

- 関数を定義するモジュール中で，__callt 宣言を行います。

```
__callt 関数宣言;
extern __callt  関数宣言;
static __callt  関数宣言;
__callt extern  関数宣言;
__callt static  関数宣言;
```

［制限］

- callt 関数は，メモリ・モデルによらず，0xC0 ～ 0xFFFF に配置します。
- 関数宣言以外に __callt を指定した場合は，コンパイル・エラーとなります。
- callt 関数の数に関するチェックは，リンク時に行います。
- 対象関数のすべての宣言に対して __callt を指定する必要があります。
  __callt の指定の有無が混在する場合は，コンパイル・エラーとなります。

```
        void func(void);
__callt void func(void);          // エラー
```

- __near と同時指定できますが，__far と同時指定した場合はコンパイル・エラーとなります。

```
__callt  void func1(void);         // 関数アドレスは near
__callt  __near void func2(void);  // 関数アドレスは near
__callt  __far void func3(void);   // エラー
```

- typedef 中に __callt を使用した場合は，コンパイル・エラーとなります。

```
typedef __callt int     CI;        // エラー
```

- 関数宣言に __callt の指定がある関数に #pragma の指定がある場合は，コンパイル・エラーとなります。

［使用例］

C ソースを以下に示します。

```
__callt void func(void){
          :
}

void main(void){
        func();    //callt 関数の呼び出し
          :
}
```

アセンブリ・ソースにおけるセクション割り付けと出力コードは以下となります。

```
        .PUBLIC _func
        .PUBLIC _main
        .PUBLIC @_func
          :

        .SECTION        .text,TEXT
_func:
          :

        .SECTION        .textf,TEXTF
_main:
        callt   [@_func]
          :

        .SECTION        .callt0,CALLT0
@_func:
        .DB2    _func
          :
```

［備考］

- キーワード __callt と #pragma callt の違いについて
  - __callt キーワードは，__far キーワードとの混在を許さずコンパイル・エラーとなります。
  - #pragma callt は，__far キーワードを付加した関数でも警告なしで __callt を指定したものとして扱います。

メモリ配置領域指定（__near/__far）

関数，変数の宣言時に，__near/__far 型修飾子を追加することにより，関数，変数の配置場所を明示的に指定することができます。

備考 1. __near も __far も付けない宣言は，メモリ・モデルにより決定するデフォルトの __near および __far に従います。

備考 2. 文中では，near，far，__near，__far は，それぞれ次の意味を持ちます。

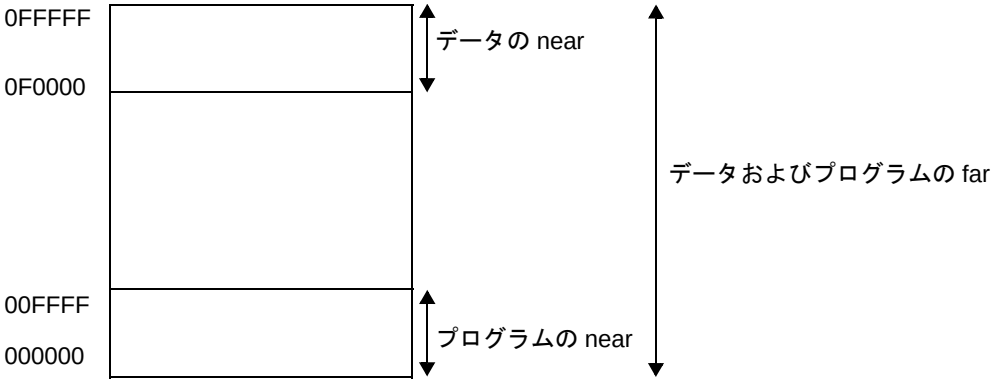
文字列	意味
near near 領域	RAM データ，および ROM データに対しては 0x0F0000 ～ 0x0FFFFF のアドレス範囲 関数に対しては 0x000000 ～ 0x00FFFF のアドレス範囲
far far 領域	RAM データ，ROM データ，関数の全てに対して，0x000000 ～ 0x0FFFFF のアドレス範囲
__near	near 領域を意味する型修飾子
__far	far 領域を意味する型修飾子

［機能］

- 型修飾子に __near と __far を追加します。変数および関数の宣言時に，__near および __far を明示することにより，コンパイラに配置先を指示します。
- __near および __far は，修飾した先の変数および関数が，以下の領域に配置されていることを意味します。

型修飾子	関数	データ
__near	0x000000 ～ 0x00FFFF	0x0F0000 ～ 0x0FFFFF
__far	0x000000 ～ 0x0FFFFF	0x000000 ～ 0x0FFFFF

図 4.15 near，far とメモリ・イメージ



- ソースコードに明示的に記述する __near および __far は，オプションによる設定よりも優先されます。
- near を指すポインタ（2 バイト），および far を指すポインタ（4 バイト）の内部表現は，ポインタ型を参照してください。
- near を指すポインタから far を指すポインタへの拡張は，以下の拡張を基本とします。

ポインタ	near から far への拡張方法
関数ポインタ	下位から 3 バイト目は 0x00 とします。最上位 1 バイトは不定です。
変数ポインタ	下位から 3 バイト目は 0x0f とします。最上位 1 バイトは不定です。

整数型などを含むキャスト全般の詳細については，キャストを参照してください。

- void へのポインタは，変数ポインタとして扱います。
- 空ポインタ定数（NULL）の内部表現は，「関数ポインタ」，「変数ポインタ」ともにゼロとします。
  far の変数ポインタは下位から 3 バイト目が 0x0f となりますが，NULL に限り 0x000000（最上位 1 バイトは不定）とします。
- __near および __far と配置セクション

	関数	初期値なし変数	初期値あり 非 const 変数	初期値あり const 変数 および文字列データ
__near	.text	.bss	.data	.const
__far	.textf	.bssf	.dataf	.constf

- 宣言
  - 宣言を右から左に見て，「変数，関数，*」から次の「*，(，宣言の左端」の間に __near/__far があれば，それを「変数，関数，*」の __near/__far 属性とします。
    もし，何もなければ「変数，関数，*」の持つデフォルトの __near/__far 属性にします。
    __near/__far は，上記の順序を変えない範囲で，ほかの宣言子と順序を変えることができます。

    例えば medium model の場合は，以下のとおりとなります。

宣言	意味
`int x;`	`int __near x;` `//int 型の変数 x は //near セクションに配置`
`int func();`	`int __far func ();` `//int 型を返す関数 func は //far セクションに配置`
`int* x;`	`int __near * __near x;` `//near を指すポインタ x は //near セクションに配置`
`int* func ();`	`int __near * __far func ();` `//near ポインタを返す関数 func は //far セクションに配置`
`int (*fp)();`	`int (__far * __near fp)();` `//far 関数ポインタ fp は //near セクションに配置`
`int __far * func ();`	`int __far * __far func();` `//far ポインタを返す関数 func は //far セクションに配置`
`int (__far * fp)();`	`int (__far * __near fp)();` `//far 関数ポインタ fp は //near セクションに配置`

  - 変数宣言に対する __near と __far の記述位置，および意味は次のとおりとします。
    __near および __far は型修飾子であり，const および volatile と同じ位置に書くことができます。__near および __far の作用対象も，const および volatile と同じです。

```
int __far i;                    //i は far セクション（.bssf）に出力
                                //sizeof(&i) は 4
__near int j;                   //j は near セクション（.bss）に出力
                                //sizeof(&j) は 2
int __far * __near p;           //p は near セクション（.bss）に出力
                                //sizeof(&p) は 2
                                //p は far セクションに存在する int オブジェクトを指す
                                //sizeof(p) は 4
void (__far * __near fp)( );    //fp は near セクション（.bss）に出力
                                //sizeof(&fp) は 2
                                //fp は far セクションに存在する関数を指す
                                //sizeof(fp) は 4
```

- 関数宣言に対する __near と __far の記述位置，及び意味は次のとおりとします。
  __near および __far は型修飾子であり，const 及び volatile と同じ位置に書くことができます。__near および __far の作用対象も，const 及び volatile と同じです。

```
void (__far func1)( );  //func1 は far セクション（.textf）に出力
                        //sizeof(&func1) は 4
void __far func2( );    //func2 は far セクション（.textf）に出力
                        //sizeof(&func2) は 4
```

- 1 つの宣言に __near および __far 型修飾子を複数回指定した場合は，エラーとなります。

```
int __near __far i;     // エラー
int __near __near i;    // エラー
int __far __far i;      // エラー
```

- 同じ変数，関数の宣言において，__near および __far が混在する場合は，エラーとなります。

```
__near int i;
__near int i;           //OK

__near int i;
__far int i;            // コンパイル・エラー

__near const int i;
       const int i;     // デフォルトが near の場合に限り OK
                        // デフォルトが far の場合はコンパイル・エラー

       const int i;
__near const int i;     // デフォルトが near の場合に限り OK
                        // デフォルトが far の場合はコンパイル・エラー
```

- 構造体・共用体メンバに __near/__far を記述した場合は，エラーとなります。【V1.04 以前】
- 構造体・共用体メンバに __near/__far を記述した場合は，警告を出力して __near/__far を無視します。【V1.05 以降】
- キーワードおよび #pragma との関係
  __near および __far を，__saddr，__callt，#pragma callt，#pragma near，#pragma far，#pragma interrupt，および #pragma interrupt_brk と同時に指定した場合の動作は，各キーワードおよび 4.2.4 #pragma 指令を参照してください。

- キャスト
  near ポインタおよび far ポインタを含むキャストは，次のとおりとします。
  - 変数ポインタの場合
    near* から far* への変換は，下位から 3 バイト目に 0x0f を追加します。
    ただし，NULL のみ near*，far* ともに 0 であるため，下位から 3 バイト目に 0x00 を追加します。
    整数型とポインタ型の変換は，値の維持を基本とします。

		変換先						
		_Bool	char	short int	near*	far*	long	long long
変換元	_Bool char	-	-	-	<a>	<b>	-	-
	short int	-	-	-	<c>	<b>	-	-
	near*	<d>	<e>	<c>	-	<f>	<g>	<g>
	far*	<d>	<e>	<e>	<e>	-	<h>	<h>
	long	-	-	-	<e>	<c>	-	-
	long long	-	-	-	<e>	<e>	-	-

<a> int に拡張してから，値を維持して near* に変換

<b> long に拡張してから，下位 3 バイトのみ値を維持して far* に変換
（理由：最上位 1 バイトは不定のため，変換の保証なし）

<c> 値を維持して変換。変換先の型が far* の場合は，型サイズを 3 バイトとして上位を切り捨て
（理由：最上位 1 バイトは不定のため，変換の保証なし）

<d> NULL の場合は 0，NULL 以外は 1 に変換

<e> 変換先の型サイズに当てはまるように上位を切り捨てる。残ったバイトの値を維持して変換。変換先の型が far* の場合は，型サイズを 3 バイトとして上位を切り捨て
（理由：最上位 1 バイトは不定のため，変換の保証なし）

<f> NULL の場合は下位から 3 バイト目に 0x00 を，NULL 以外は下位から 3 バイト目に 0x0F を設定
（理由：最上位 1 バイトは不定のため，変換の保証なし）

<g> far ポインタに拡張 <f> してから，上位バイト[注]（far ポインタの不定部分を含む）をゼロで拡張

<h> 上位バイト[注]（far ポインタの不定部分を含む）をゼロで拡張

注 long 型の場合は上位 1 バイト，long long 型の場合は上位 5 バイトとなります。

- 関数ポインタの場合
near* から far* への変換は，下位から 3 バイト目に 0x00 を追加します。
far* から near* への変換は，上位 2 バイトを切り捨てます。
ポインタからポインタへの変換以外は，変数ポインタと同じとします。

- 変数ポインタと関数ポインタ間の変換

  - オプション -strict_std 使用時は，変数ポインタと関数ポインタの型変換はエラーとなります。暗黙変換だけでなく，明示的型変換もエラーとなります。

  - オプション -strict_std 不使用時は，変数ポインタと関数ポインタの型変換は警告を出して変換します。
near/far が同一の場合は，サイズが同一であるため，型を変えるだけで操作はしません。
far から near への変換は，上位 2 バイトを破棄して切り縮めます。
near から far への変換は，同じ型のまま near から far に拡張した後に，型を変換します。

- キャストの例

  - ポインタ間の代入

```
char __near* o_np;
char __far* o_fp;

typedef void(FT)(void);

__near FT* f_np;
__far FT* f_fp;

void func(){
    o_fp = o_np;    //o_fp の上位 / 下位 2 バイト =(0xnn00, 0x0000) or (0xnn0f, o_np)
    f_fp = f_np;    //f_fp の上位 / 下位 2 バイト =(0xnn00, f_np)
}
```

  - 整数定数から “ 変数ポインタ ” への変換

```
(char __near*)(char)0x12;           //0x0012
(char __near*)0x34;                 //0x0034
(char __far*)0x56;                  //0x00000056
(char __far*)(char __near*)0x78;    //0x000f0078
```

- 整数定数から “ 関数ポインタ ” への変換

```
typedef void(FT)(void);

void func1(){
  (__far FT*)0x34;                //0x00000034
  (__far FT*)(char __near*)0x56;  //0x000f0056
                                  //(char __near*)->(char __far*)->(__far FT*)
  (__far FT*)(char __far*)0x78;   //0x00000078
  (__far FT*)(__near FT*)0xab;    //0x000000ab
  (__far FT*)(__far FT*)0xcd;     //0x000000cd
}
```

- 変数から “ 関数ポインタ ” への変換

```
typedef void(FT)(void);

void func2(__near FT* f_np, unsigned int i, unsigned char ch){
        (__far FT*)f_np;        //0xnn00, f_np
        (__far FT*)i;           //0xnn00, i
        (__far FT*)ch;          //0xnn00, ch（符号なし 2 バイト）
}
```

- 変数からポインタへの変換

```
signed char sc;
signed short ss;
signed long sl;

unsigned char uc;
unsigned short us;
unsigned long ul;

void func3(){
        (char __near*)uc;       //0x00, uc
        (char __near*)sc;       //(0x00 or 0xff), sc

        (char __far*)uc;        //nn, 0x00, 0x00, uc
        (char __far*)us;        //nn, 0x00, us(1), us(0)
        (char __far*)ul;        //nn, ul(2), ul(1), ul(0)

        (char __far*)sc;        //nn, (0x0000 or 0xffff), sc
        (char __far*)ss;        //nn, (0x00 or 0xff), ss(1), ss(0)
        (char __far*)sl;        //nn, sl(2), sl(1), sl(0)

        (__far FT*)uc;          //nn, 0x00, 0x00, uc
        (__far FT*)us;          //nn, 0x00, us(1), us(0)
        (__far FT*)ul;          //nn, ul(2), ul(1), ul(0)

        (__far FT*)sc;          //nn, (0x0000 or 0xffff), sc
        (__far FT*)ss;          //nn, (0x00 or 0xff), ss(1), ss(0)
        (__far FT*)sl;          //nn, sl(2), sl(1), sl(0)
}
```

- ポインタから変数への変換

```
char __near* o_np;
char __far* o_fp;

typedef void(FT)(void);

__near FT* f_np;
__far FT* f_fp;

signed char sc;
signed short ss;
signed long sl;

unsigned char uc;
unsigned short us;
unsigned long ul;

void func(){
        uc = o_np;  //o_np の下位 1 バイト
        uc = o_fp;  //o_fp の下位 1 バイト
        uc = f_np;  //f_np の下位 1 バイト
        uc = f_fp;  //f_fp の下位 1 バイト

        us = o_np;  //o_np のビット・パターン維持
        us = o_fp;  //o_fp の下位 2 バイト
        us = f_np;  //f_np のビット・パターン維持
        us = f_fp;  //f_fp の下位 2 バイト

        ul = o_np;  //(0x0000,o_np)or(0x000f,o_np)
        ul = o_fp;  //(0x00,o_fp の下位 3 バイト，o_fp の下位 2 バイト )
        ul = f_np;  //(0x0000,f_np)
        ul = f_fp;  //(0x00,f_fp の下位 3 バイト，f_fp の下位 2 バイト )

        sc = o_np;  //o_np の下位 1 バイト
        sc = o_fp;  //o_fp の下位 1 バイト
        sc = f_np;  //f_np の下位 1 バイト
        sc = f_fp;  //f_fp の下位 1 バイト

        ss = o_np;  //o_np のビット・パターン維持
        ss = o_fp;  //o_fp の下位 2 バイト
        ss = f_np;  //f_np のビット・パターン維持
        ss = f_fp;  //f_fp の下位 2 バイト

        sl = o_np;  //(0x0000,o_np)or(0x000f,o_np)
        sl = o_fp;  //(0x00,o_fp の下位 3 バイト，o_fp の下位 2 バイト )
        sl = f_np;  //(0x0000,f_np)
        sl = f_fp;  //(0x00,f_fp の下位 3 バイト，f_fp の下位 2 バイト )
}
```

- ポインタ演算

  - far ポインタの加算は下位 2 バイトで行ないます。上位は変化しません。

```
char __far* ptr = (char __far*)0x5ffff + 1;     //0x00050000
```

  - far ポインタの減算は下位 2 バイトで行ないます。上位は変化しません。

```
char __far* ptr = (char __far*)0x050002 - 3;    //0x05ffff
```

- near ポインタと far ポインタ間の減算は，右項の型を左項の型にキャストしてから減算します。

```
__near int i;
__far int j;

void func( )
{
        &j - &i;        //OK (&j) - ((int __far *)(&i))
        &i - &j;        //OK (&i) - ((int __near *)(&j))

        &j - (__far int*)&i; //OK
}
```

- near ポインタと整数の演算結果は，near ポインタとします。

```
int b = ((char __near *)0xffff+1 == (char __near *)0);  //=1（真）
```

- ポインタ比較
  - オプション -strict_std 指定時は，ポインタと整数間で直接比較できません。しかし，オプション -strict_std 非指定時は，警告を出して直接比較を可能とします。
    警告時の比較は，整数型をポインタ型に合わせて比較します。整数型からポインタ型への変換は，キャストの項目に書いた規則に従います。
  - オプション -strict_std 指定時は，変数ポインタと関数ポインタで直接比較できません。しかし，オプション -strict_std 非指定時は，警告を出して直接比較を可能とします。
    警告時の比較は，次のように機能します。

  <a> 右辺の型を，左辺の型にキャストします。

  <b> 両辺の型の near および far が異なる場合は，near ポインタから far ポインタへのキャストが発生します。

```
obj_near_ptr  == func_near_ptr   →   obj_near_ptr == (obj *)func_near_ptr
func_near_ptr == obj_near_ptr    →   func_near_ptr == (func *)obj_near_ptr
obj_near_ptr  == func_far_ptr
                      →   (obj __far *)obj_near_ptr == (obj __far *)func_far_ptr
func_f_ptr == obj_n_ptr          →   func_f_ptr == (func _far *)obj_n_ptr
func_n_ptr == obj_f_ptr
                      →   (func __far *)func_n_ptr == (func __far *)obj_f_ptr
obj_f_ptr == func_n_ptr          →   obj_f_ptr == (obj __far *)func_n_ptr
```

  - far ポインタの等値演算 (==,!=) は，下位 3 バイトで行ないます。最上位の 1 バイトは演算結果に影響しません。

```
if((char __far*)0x1105ffffUL == (char __far*)0x05ffffUL)
        OK();
else
        NG();
```

  - far ポインタの関係演算 (<,>,<=,>=) は，下位 2 バイトのみで比較します。上位の 2 バイトは演算結果に影響しません。
    上位バイトを含めて比較する場合は，unsigned long にキャストしてから比較する必要があります。
- ptrdiff_t の型
  near ポインタ，far ポインタの演算に関わらず常に signed int 型（2 バイト）とします。

［方法］

- 関数，変数の宣言時に，__near/__far 型修飾子を追加します。

［使用例］

```
__near int i1;                          (1)
__far int i2;                           (2)
__far int *__near p1;                   (3)
__far int *__near *__far p2;            (4)
__far int func1( );                     (5)
__far int *__near func2 ( );            (6)
int (__near *__far fp1 ) ( );           (7)
__far int * (__near *__near fp2 ) ( );  (8)
__near int * (__far *__near fp3 ) ( );  (9)
__near int * (__near *__far fp4 ) ( );  (10)
```

(1) i1 は int 型で，near 領域に配置

(2) i2 は int 型で，far 領域に配置

(3) p1 は “far 領域にある int 型 ” を指す 4 バイト型の変数で，変数自身は near 領域に配置

(4) p2 は「near 領域にある，“far 領域にある int 型 ” を指す 4 バイト型」を指す 2 バイト型の変数で，変数自身は far 領域に配置

(5) func1 は “int 型 ” を返す関数で，関数自身は far 領域に配置

(6) func2 は「“far 領域にある int 型 ” を指す 4 バイト型」を返す関数で，関数自身は near 領域に配置

(7) fp1 は「“int 型 ” を返す near 領域にある関数」を指す 2 バイト型の変数で，変数自身は far 領域に配置

(8) fp2 は “「“far 領域にある int 型 ” を指す 4 バイト型」を返す near 領域にある関数 ” を指す 2 バイト型の変数で，変数自身は near 領域に配置

(9) fp3 は “「“near 領域にある int 型 ” を指す 2 バイト型」を返す far 領域にある関数 ” を指す 4 バイト型の変数で，変数自身は near 領域に配置

(10) fp4 は “「“near 領域にある int 型 ” を指す 2 バイト型」を返す near 領域にある関数 ” を指す 2 バイト型の変数で，変数自身は far 領域に配置

インライン関数指定（__inline）

インライン関数であることをコンパイラに示唆します。

［機能］

- コンパイラは，可能であれば __inline 宣言した関数をインライン展開します。
- 同一翻訳単位内で，同じ関数に対して #pragma noinline と __inline を同時に指定する場合はエラーとします。
- オプション -Oinline_level により，インライン展開対象となる関数の条件が変化します。
  オプションの影響については「2.5.1 コンパイル・オプション」の -Oinline_level の説明を参照してください。
- __inline 宣言した関数は，その関数を呼び出したところにインライン展開します。
- 指定した関数の呼び出しと定義の間で下記のいずれかに当てはまる場合，警告メッセージが出力されてインライン指定は無視されます。
  - 引数の数が異なる場合
  - 戻り値の型や引数の型が異なり，型変換が出来ない場合
    型変換が可能な場合には，型を変換してインライン展開します。
    なお，変換する場合には，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ます。
    ただし，-strict_std オプション指定時はエラーとなります。
- 指定した関数が次のいずれかに当てはまる場合，インライン指定は無視されます。なお，メッセージは出力されません。
  - 可変個引数の関数
  - 処理中に自分自身を呼び出す関数
  - 展開対象関数のアドレスを介して呼び出しを行っている場合

［効果］

- 関数の呼び出しと復帰コードが無くなるため，実行速度が速くなります。
- インライン展開されたコードが，関数呼び出し前後のコードと合わせて最適化対象になるため，最適化の効果が大きくなる場合があります。
- インライン展開の回数が多いと，コードサイズが大きくなる可能性があります。

［方法］

- 関数の定義に，__inline を追加します。

［使用例］

```
extern int      gi;
__inline int    i_func(int i)
{
        return ++i;
}

void func(int j)
{
        gi = i_func(j);
}
```

［注意］

- __inline は，インライン展開されることを保証するものではありません。コンパイル時間やメモリ使用量の増大を考慮した制限により，インライン展開を抑止する場合があります。
- -lang=c99 オプションの指定時，キーワード __inline はキーワード inline の別名とします。すなわち C99 のインライン仕様に従います。

セクション・アドレス演算子（__sectop/__secend）

セクション・アドレス演算子です。

［機能］

- __sectop で指定した*セクション名*の先頭アドレスを参照します。
- __secend で指定した*セクション名*の末尾 +1 アドレスを参照します。
- __sectop( )，および __secend( ) の戻り型は void __far* とします。
- __sectop，または __secend で取得した値と定数との加減算を記述してはいけません。

［方法］

```
__sectop(" セクション名 ")
__secend(" セクション名 ")
```

［使用例］

C ソースを以下に示します。

```
const struct {
        void __far *rom_s;
        void __far *rom_e;
} DTBL[]={__sectop("XXX"), __secend("XXX")};
```

アセンブリ・ソースにおける宣言は以下となります。

```
        .PUBLIC         _DTBL
        .SECTION        .const, CONST
        .ALIGN          2
_DTBL:
        .DB2    LOWW(STARTOF(XXX))                      ;rom_s の下位 2 バイト
        .DB     LOW(HIGHW(STARTOF(XXX)))                ;rom_s の上位 1 バイト
        .DB     0x00
        .DB2    LOWW(STARTOF(XXX)+SIZEOF(XXX))          ;rom_e の下位 2 バイト
        .DB     LOW(HIGHW(STARTOF(XXX)+SIZEOF(XXX)))    ;rom_e の上位 1 バイト
        .DB     0x00
```

ハードウェア割り込みハンドラ（#pragma interrupt）

ハードウェア割り込みに対応したオブジェクト・コードを出力します。

［機能］

- 割り込みベクタが生成されます。
- 対象の関数定義をハードウェア割り込みハンドラとして，RETI による復帰コードが生成されます。
- 汎用レジスタの退避領域としてスタック，またはレジスタ・バンクが指定できます。
- 割り込みハンドラ定義は通常関数同様に，.text セクション，または .textf セクションに出力されます。セクション名は，#pragma section により変更可能です。
  ただし，ベクタテーブルを指定した場合は，割り込みハンドラの先頭アドレスは 16 ビットでアドレッシングできなければなりません。
- ベクタテーブルを指定した場合，明示的，暗黙問わず __far を指定しても強制的に __near 指定となります。
  警告は出力しません。
- ベクタテーブルを指定しなかった場合，明示的，暗黙問わず関数の __near/__far 指定に従います。

［効果］

- C ソース・レベルで割り込みハンドラの記述が可能となります。

［方法］

- #pragma 指令に関数名，割り込み仕様を指定します。
- #pragma 指令は関数定義の前に書きます。

```
#pragma interrupt   [(] 割り込みハンドラ名 [( 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。

項目	形式	内容
ベクタテーブル指定	vect= *アドレス*	割り込みハンドラの先頭アドレスを格納するベクタテーブルのアドレス。 省略時はベクタテーブルを生成しません。 *アドレス* :2，8，10 または 16 進数定数 0x0 ～ 0x7c の偶数のみ（範囲外はエラーとする）
レジスタ・バンク指定	bank={RB0|RB1|RB2|RB3}	割り込みハンドラ内で使用するレジスタ・バンク。[注 1] 割込みハンドラの実行に指定のバンク中のレジスタを使います。 ES と CS はスタックに退避します。 汎用レジスタの退避および復帰をレジスタ・バンクを切り替える SEL 命令で実現するため，コードサイズを小さくすることができます。 本指定を省略した場合，汎用レジスタはスタックに退避します。[注 2]
多重割り込み許可指定	enable={true|false}	true : 関数入口での多重割り込みを可能とします。 すなわち，レジスタの退避処理よりも前に EI を生成します。 false : EI を生成しません。enable を省略した場合は，EI を生成しません。

注 1. 割り込みハンドラを呼び出す前に使用しているレジスタ・バンクとは，異なるレジスタ・バンクを指定してください。同じレジスタ・バンクが指定された場合には，退避したレジスタの内容を復帰できなくなります。

注 2. 割り込み時のレジスタ退避やスタックフレームの詳細は「9.1 関数呼び出しインタフェース」を参照してください。

同一の項目を，同時に複数回書く場合はコンパイル・エラーとなります。
ただし，ベクタテーブルの指定は，アドレスが重複しなければ複数記述可能とします。【V1.06 以降】

```
#pragma interrupt   func(vect=2,vect=2) // エラー
```

```
#pragma interrupt   func(vect=2,vect=8) // OK
```

```
#pragma interrupt   func(vect=2)
#pragma interrupt   func(vect=8)        // OK
```

「vect= アドレス」指定がある場合はベクタテーブルを生成します。このため，スタートアップ・ルーチンなどにアセンブリ言語のセクション定義疑似命令でベクタテーブルを定義すると，リンク時にエラーとなります。
「vect= アドレス」指定を記述した場合，それ以外にアセンブリ言語でベクタテーブルを記述するには，セクション定義疑似命令ではなく .VECTOR 疑似命令で記述してください。

［制限］

- 割り込みハンドラを通常関数のように呼び出すと，コンパイル・エラーとなります。
- __inline，__callt，および他の #pragma を指定すると，コンパイル・エラーとなります。
- 割り込みハンドラは，引数，戻り型ともに void 型で宣言しなければなりません（例：void func (void);）。
  void 型でない場合はコンパイル・エラーとなります。
- 割り込みハンドラ中にレジスタ利用や関数呼び出しがない場合，#pragma interrupt でレジスタ・バンク切り替え指定をしてもレジスタ・バンク切り替え命令は出力しません。
  これが発生する例として以下があります。
  - 割り込みハンドラ中に，SFR に定数値を設定するなどのレジスタを使わない命令のみが出力される場合

［使用例］

(1) 割り込みハンドラ中に関数呼び出しがない場合

(a) デフォルト指定の場合
【入力プログラム】

```
#include “iodefine.h”                    /*iodefine.h をインクルードすると */
#pragma interrupt   inter (vect=INTP0)  /*vect 指定に割り込み要因名が使える */

void inter ( void ) {
        /* 割り込み処理（AX，HL，ES のみ使用）*/
}
```

【出力プログラム】

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ; ベクタテーブル指定があるため __near 指定とみなす
_inter:
        push    AX              ; 使用する汎用レジスタの，スタックへの退避コード
        push    HL
        mov     A, ES
        push    AX

        ;INTP0 端子入力に対する割り込み処理（関数本体。AX，HL，ES のみ使用）

        pop     AX              ; 使用する汎用レジスタの，スタックからの復帰コード
        mov     ES, A
        pop     HL
        pop     AX
        reti
```

(b) レジスタ・バンク指定がある場合

【入力プログラム】

```
#pragma interrupt   inter (vect=INTP0, bank=RB1)

void inter ( void ) {
        /* 割り込み処理（ES は使用，CS は使用しない）*/
}
```

【出力プログラム】

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ; ベクタテーブル指定があるため __near 指定とみなす
_inter:
        sel     RB1             ; レジスタ・バンクの切り替えコード
        mov     A, ES           ; 使用する ES，CS レジスタのスタックへの退避コード
        push    AX

        ;INTP0 端子入力に対する割り込み処理（関数本体。ES は使用，CS は使用しない）

        pop     AX              ; 使用する ES，CS レジスタのスタックからの復帰コード
        mov     ES, A
        reti
```

(2) 割り込みハンドラ中に関数呼び出し（#pragma inline_asm 宣言された関数の関数呼び出しを含む）がある場合

(a) デフォルト指定の場合

【入力プログラム】

```
#pragma interrupt   inter (vect=INTP0)

void inter ( void ) {
        /* 割り込み処理 */
}
```

【出力プログラム】

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ; ベクタテーブル指定があるため __near 指定とみなす
_inter:
        push    AX
        push    BC
        push    DE
        push    HL
        mov     A, ES           ;ES，CS，および全汎用レジスタのスタックへの退避コード
        mov     X, A
        mov     A, CS
        push    AX

        ;INTP0 端子入力に対する割り込み処理（関数本体）

        pop     AX              ;ES，CS，および全汎用レジスタのスタックからの復帰コード
        mov     CS, A
        mov     A, X
        mov     ES, A
        pop     HL
        pop     DE
        pop     BC
        pop     AX
        reti
```

(b) レジスタ・バンク指定がある場合

【入力プログラム】

```
#pragma interrupt   inter (vect=INTP0, bank=RB1)

void inter ( void ) {
        /* 割り込み処理 */
}
```

【出力プログラム】

```
_inter  .vector 0x0008          ;INTP0
        .section    .text, TEXT ; ベクタテーブル指定があるため __near 指定とみなす
_inter:
        sel     RB1             ; レジスタ・バンクの切り替えコード
        mov     A, ES           ;ES，CS レジスタのスタックへの退避コード
        mov     X, A
        mov     A, CS
        push    AX

        ;INTP0 端子入力に対する割り込み処理（関数本体）

        pop     AX              ;ES，CS レジスタのスタックからの復帰コード
        mov     CS, A
        mov     A, X
        mov     ES, A
        reti
```

ソフトウェア割り込みハンドラ（#pragma interrupt_brk）

ソフトウェア割り込みに対応したオブジェクト・コードを出力します。

［機能］

- 割り込みベクタが生成されます。
- 対象の関数定義をソフトウェア割り込みハンドラとして，RETB による復帰コードが生成されます。
- 割り込みハンドラ定義は通常関数同様に，.text セクションに出力されます。セクション名は，#pragma section により変更可能です。
  ただし，割り込みハンドラの先頭アドレスは 16 ビットでアドレッシングできなければなりません。
- 明示的，暗黙問わず __far を指定しても強制的に __near 指定となります。
  警告は出力しません。

［効果］

- C ソース・レベルで割り込みハンドラの記述が可能となります。

［方法］

- #pragma 指令に関数名，割り込み仕様を指定します。
- #pragma 指令は関数定義の前に書きます。

```
#pragma interrupt_brk   [(] 割り込みハンドラ名 [( 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。

項目	形式	内容
レジスタ・バンク指定	bank={RB0|RB1|RB2|RB3}	割り込みハンドラ内で使用するレジスタ・バンク。[注 1] 割込みハンドラの実行に指定のバンク中のレジスタを使います。 ES と CS はスタックに退避します。 汎用レジスタの退避および復帰をレジスタ・バンクを切り替える SEL 命令で実現するため，コードサイズを小さくすることができます。 本指定を省略した場合，汎用レジスタはスタックに退避します。[注 2]
多重割り込み許可指定	enable={true|false}	true：関数入口での多重割り込みを可能とします。 つまり，コンパイラが生成する割り込みハンドラの先頭に EI を生成します。 false：EI を生成しません。enable を省略した場合は，EI を生成しません。

注 1. 割り込みハンドラを呼び出す前に使用しているレジスタ・バンクとは，異なるレジスタ・バンクを指定してください。同じレジスタ・バンクが指定された場合には，退避したレジスタの内容を復帰できなくなります。

注 2. 割り込み時のレジスタ退避やスタックフレームの詳細は「9.1 関数呼び出しインタフェース」を参照してください。

同一の項目を，同時に複数回書く場合はコンパイル・エラーとなります。

例

```
#pragma interrupt_brk func(bank=RB0, bank=RB1) // エラー
```

本 #pragma はベクタテーブルを生成します。このため，スタートアップ・ルーチンなどにアセンブリ言語のセクション定義疑似命令でベクタテーブルを定義していると，リンク時にエラーとなります。
アセンブリ言語でもベクタテーブルを記述するには，セクション定義疑似命令ではなく .VECTOR 疑似命令で記述してください。

［制限］

- 割り込みハンドラを通常関数のように呼び出すと，コンパイル・エラーとなります。
- __inline，__callt，および他の #pragma を指定すると，コンパイル・エラーとなります。
- 割り込みハンドラは，引数，戻り型ともに void 型で宣言しなければなりません（例：void func (void);）。
  void 型でない場合はコンパイル・エラーとなります。
- 割り込みハンドラ中にレジスタ利用や関数呼び出しがない場合，#pragma interrupt でレジスタ・バンク切り替え指定をしてもレジスタ・バンク切り替え命令は出力しません。

［使用例］

割り込み要因が BRK であり，復帰コードに RETB を使うことを除けば，出力プログラムは #pragma interrupt と同じとします。
よって，割り込みハンドラ中に関数呼び出しがない場合の使用例を示します。

【入力プログラム】

```
#pragma interrupt_brk   inter

void __near inter ( void ) {
        /* 割り込み処理（AX，HL，ES のみ使用）*/
}
```

【出力プログラム】

```
_inter  .vector 0x007E          ;BRK
        .SECTION    .text, TEXT
_inter:
        push    AX              ; 使用する汎用レジスタの，スタックへの退避コード
        push    HL
        mov     A, ES
        push    AX

        ;BRK 命令に対する割り込み処理（関数本体。AX，HL，ES のみ使用）

        pop     AX              ; 使用する汎用レジスタの，スタックからの復帰コード
        mov     ES, A
        pop     HL
        pop     AX
        retb
```

コンパイラ出力セクション名の変更（#pragma section）

コンパイラが出力するセクション名を切り替えます。

［機能］

- コンパイラが出力するセクション名を切り替えます。
  コンパイラが出力するセクション名とデフォルトの配置については，「6.1.1 セクション名」を参照してください。
- #pragma 指令以降に書いた宣言に作用します。

［効果］

- 特定のデータや関数の集まりに対してデフォルトとは異なるセクション名を付けることで，それらのデータや関数の集まりをほかのデータや関数とは別のアドレスに配置したり，セクション単位で特定の処理をすることが可能となります。

［方法］

```
#pragma section セクション種別 変更セクション名
        セクション種別:{text|const|data|bss}
```

指定した種別のセクションの名前を変更します。

```
#pragma section 変更セクション名
```

すべての種別のセクションの名前を変更します。

```
#pragma section
```

すべての種別のセクションの名前をデフォルト・セクション名にします。

- 変更セクション名に使用可能な文字は，次のとおりとします。
  ただし，先頭は 0 ～ 9 以外の文字のみ使用可能とします。
  また “.” はセクション種別を指定する場合のみ，先頭のみ使用可能であり，先頭を除く文字列内で使用した場合はコンパイル・エラーとなります。セクション種別を指定しない場合は，“.” を使用するとコンパイル・エラーとなります。
  - 0 ～ 9
  - a ～ z
  - A ～ Z
  - .
  - @
  - _
- セクション種別が text であればその #pragma 宣言以降に定義された関数のセクション名を変更します。
  ただし，割り込みハンドラの先頭アドレスは 16 ビットでアドレッシングできなければなりません。
- セクション種別が const，data，または bss の場合は，その #pragma 宣言以降に実体を定義した const，data，または bss のセクション名を変更します。
- セクション種別と変更セクション名の両方を指定する場合，セクション名には以下の規則で文字列を付加します。
  - near セクション (.text, .const, .data, .bss)
    変更セクション名 + "_n"

```
#pragma section data MyData
int __near ndata = 5;           // 生成セクション名 : MyData_n
```

- far セクション (.textf, .constf, .dataf, .bssf)
  変更セクション名 + "_f"

```
#pragma section bss MyBss
int __far fdata;                // 生成セクション名 : MyBss_f
```

- saddr セクション (.sdata, .sbss)
  変更セクション名 + "_s"

```
#pragma section bss MyBss
int __saddr sdata;              // 生成セクション名 : MyBss_s
```

	#pragma section text MyText	#pragma section bss MyBss	#pragma section data MyData	#pragma section const MyConst
void __near func() { }	MyText_n			
void __far func() { }	MyText_f			
int __near i;		MyBss_n		
int __far i;		MyBss_f		
int __saddr i;		MyBss_s		
int __near i = 5;			MyData_n	
int __far i = 5;			MyData_f	
int __saddr i = 5;			MyData_s	
const int __near i = 5;				MyConst_n
const int __far i = 5;				MyConst_f

- 変更セクション名のみを指定する場合，その #pragma 宣言以降にあるプログラム領域，定数領域，初期化データ領域，および未初期化データ領域のすべてのセクション名を変更します。この場合，各セクションの変更後のセクション名は，現在のセクション名の後に変更セクション名の文字列を追加し，更に _s/_n/_f のいずれかを付けたセクション名となります。現在のセクション名に _n/_f が付いている場合は，それを取り除きます。

```
#pragma section bss     XXX
int __near i;                   // セクション名は XXX_n
#pragma section YYY
int __far j;                    // セクション名は XXXYYY_f
                                //bss は near/far まとめてセクション名が変化するため，
                                //__far でも XXX が付加される。
```

- セクション種別も変更セクション名も記述しなかった場合は，その #pragma 宣言以降にあるプログラム領域，定数領域，初期化データ領域，および未初期化データ領域の全てのセクション名をデフォルト・セクション名に戻します。
- #pragma section を関数内に記述した場合，有効範囲は関数内に限らず記述位置以降すべてとなります。関数内で text のセクションを変更した場合，次の関数から有効となります。switch テーブルのセクション配置は，各関数の先頭時点で指定した const データのセクションとなります。

［制限］

- 割り込みベクタテーブルはセクション名を変更できません。

［使用例］

(1) セクション名とセクション種別を指定する場合

```
int __near ni_1;        //.bss
int __far fi_1;         //.bssf

#pragma section bss MyBss
int __near ni_2;        //MyBss_n
int __far fi_2;         //MyBss_f

#pragma section
int __near ni_3;        //.bss
int __far fi_3;         //.bssf
```

(2) セクション種別を省略する場合

```
#pragma section abc
int __near na;                  //.bssabc_n セクションに配置
int __far fa;                   //.bssfabc_f セクションに配置
int __near ni=1;                //.dataabc_n セクションに配置
int __far fi=1;                 //.datafabc_f セクションに配置
const int __near nc=1;          //.constabc_n セクションに配置
const int __far fc=1;           //.constfabc_f セクションに配置
void __near f(void)             //.textabc_n セクションに配置
{
        na=nc;
}

#pragma section
int __near nb;                  //.bss セクションに配置
void __near g(void)             //.text セクションに配置
{
        nb=nc;
}
```

(3) セクション種別指定なしとありの混在の場合

```
int __near ni_1;                //.bss
int __far fi_1;                 //.bssf

#pragma section bss MyBss
int __near ni_2;                //MyBss_n
int __far fi_2;                 //MyBss_f

#pragma section XXX
int __near ni_3;                //MyBssXXX_n セクションに配置
int __far fi_3;                 //MyBssXXX_f セクションに配置

#pragma section
int __near ni_4;                //.bss
int __far fi_5;                 //.bssf
```

RTOS 用割り込みハンドラ（#pragma rtos_interrupt）

RL78 向けルネサス RTOS 用の割り込みハンドラを記述することができます。

［機能］

- #pragma rtos_interrupt 指令で指定された関数名を RL78 向けルネサス RTOS 用割り込みハンドラと解釈します。
- ベクタテーブル指定時は，記述された関数名のアドレスを，指定された割り込みベクタテーブルに登録します。
- 割り込みハンドラの本体（関数定義）部分のコードは，.text セクション，または .textf セクションに出力されます。
  セクション名は，#pragma section により変更可能です。
  ただし，ベクタテーブル指定時は，割り込みハンドラの先頭アドレスは 16 ビットでアドレッシングできなければなりません。
- ベクタテーブルを指定した場合，明示的，暗黙問わず __far を指定しても強制的に __near 指定となります。
  警告は出力しません。
- ベクタテーブルを指定しなかった場合，明示的，暗黙問わず関数の __near/__far 指定に従います。
- RTOS 用割り込みハンドラは，次の順番でコード生成を行います。
  (a) call !!addr20 命令によるカーネル・シンボル __kernel_int_entry の呼び出し
  ベクタテーブル指定時は，__kernel_int_entry には割り込みアドレスを実引数として渡します。
  ベクタテーブル未指定時は，__kernel_int_entry には実引数を渡しません。
  (b) ローカル変数領域の確保（ローカル変数があるときのみ）
  (c) 関数本体の実行
  (d) ローカル変数領域の解放（ローカル変数があるときのみ）
  (e) ラベル __kernel_int_exit に br !!addr20 命令で無条件分岐

［効果］

- C ソース・レベルで，RTOS 用割り込みハンドラの記述が可能となります。

［方法］

- 次の #pragma 指令により，割り込みアドレスと関数名を指定します。

```
#pragma rtos_interrupt   [(] 関数名 [(vect= アドレス注 )][)]
```

注 *アドレス* : 2，8，10 または 16 進数定数
定数値は 0 ～ 0x7c の偶数のみ（範囲外はエラー）

- #pragma 指令は関数定義の前に書いてください。

［制限］

- #pragma rtos_interrupt 指定した関数を通常関数のように呼び出すと，コンパイル・エラーとなります。

```
#pragma rtos_interrupt func (vect=8)
void func(void) {}

void xxx()
{
        func();                 // エラー
}
```

- 関数は，引数，戻り型ともに void 型で宣言しなければなりません（例：void func (void);）。
  void 型でない場合はコンパイル・エラーとなります。
- RTOS 用割り込みハンドラに __inline，__callt および他 #pragma を指定すると，コンパイル・エラーとなります。

- #pragma rtos_interrupt の宣言行以降は，関数，または変数として _kernel_int_exit および _kernel_int_entry の呼び出しと定義はエラーとなります。

```
#pragma rtos_interrupt func (vect=8)
void func(void) {
        _kernel_int_entry();    // エラー
        _kernel_int_exit();     // エラー
}

void    _kernel_int_entry(){}   // エラー
void    _kernel_int_exit(){}    // エラー
```

［使用例］

ベクタテーブルの指定があり，割り込みハンドラ中に関数呼び出しがない場合の使用例を示します。

【入力プログラム】

```
#include "iodefine.h"

#pragma rtos_interrupt inthdr (vect=INTP0)
volatile int g;

void inthdr(void) {
        volatile int a;
        a = 1;
        g = 1;
}
```

【出力プログラム】

```
        .SECTION .textf,TEXTF
_inthdr .vector 0x0008
_inthdr:
        push    ax                      ;ax レジスタの退避
        movw    ax, #0x0008
        call    !!__kernel_int_entry    ;ax 以外のレジスタの退避
        push    hl                      ; ローカル変数の領域確保
        onew    ax
        movw    [sp+0x00], ax
        movw    !LOWW(_g), ax
        pop     hl                      ; ローカル変数の領域解放
        br      !!__kernel_int_exit     ; 割り込み元タスクの実行再開時に
                                        ; 全レジスタを復帰するよう処理
```

ベクタテーブルの指定がなく，割り込みハンドラ中に関数呼び出しがない場合の使用例を示します。

【入力プログラム】

```
#include "iodefine.h"

#pragma rtos_interrupt inthdr
volatile int g;

void inthdr(void) {
        volatile int a;
        a = 1;
        g = 1;
}
```

【出力プログラム】

```
        .SECTION .textf,TEXTF
_inthdr:
        call    !!__kernel_int_entry    ;ax 以外のレジスタの退避
        push    hl                      ; ローカル変数の領域確保
        onew    ax
        movw    [sp+0x00], ax
        movw    !LOWW(_g), ax
        pop     hl                      ; ローカル変数の領域解放
        br      !!__kernel_int_exit     ;割り込み元タスクの実行再開時に
                                        ;全レジスタを復帰するよう処理
```

RTOS 用タスク関数（#pragma rtos_task）

#pragma 指令により，指定された関数を RL78 向けルネサス RTOS 用のタスクと解釈します。

［機能］

- #pragma rtos_task で指定された関数名を RTOS 用のタスクと解釈します。
- タスク関数は，引数なし，または signed long 型の 1 引数を使用可能とし，戻り値の記述はできません。
  2 つ以上の引数を記述した場合，signed long 型でない引数を記述した場合，または戻り値を記述した場合，エラーとなります。
- RTOS 用タスク関数では関数出口処理を出力しません。
- RTOS 用タスク関数の最後で，常に RTOS のサービス・コール ext_tsk を呼びます。
- RTOS サービス・コール ext_tsk は，br !!addr20 命令で対応する OS 内部関数を呼びます。通常関数の最後で ext_tsk を発行した場合は，関数出口処理を出力しません。
- タスク関数の本体（関数定義）部分のコードは，.text セクション，または .textf セクションに出力されます。セクション名は，#pragma section により変更可能です。

［効果］

- C ソース・レベルで，RTOS 用タスク関数を記述することができます。
- 関数出口処理が出力されなくなるため，コード効率が良くなります。

［方法］

- 次の #pragma 指令に，関数名を指定します。

```
#pragma rtos_task   [(] タスク関数名 [ ,...][)]
```

［制限］

- RTOS 用タスク関数は __inline および他 #pragma を指定すると，コンパイル・エラーとなります。
- RTOS 用タスク関数を通常の関数のように呼び出すと，コンパイル・エラーとなります。

［使用例］

C ソースを以下に示します。

```
#pragma rtos_task       func1
#pragma rtos_task       func2
extern void ext_tsk(void);
extern void g(int *a);

void func1 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        ext_tsk( );
}

void func2 ( signed long x ) {
        int a[3];
        a[0] = 1;
        g(a);
}

void func3 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        ext_tsk( );
}

void func4 ( void ) {
        int a[3];
        a[0] = 1;
        g(a);
        if ( a[0] )
                ext_tsk( );
}
```

コンパイラの出力アセンブリ・ソースは，以下のようになります。

```
    .section        .textf, TEXTF
_func1 :
    subw    sp, #0x06               ; スタックの確保
    onew    ax
    movw    [sp+0x00], ax           ; 配列 a の要素への代入
    movw    ax, sp
    call    !!_g
    call    !!_ext_tsk              ;ext_tsk 関数の呼び出し
    br      !!__kernel_task_exit    ; タスク関数が常に出力する __kernel_task_exit 呼び出し
                                    ; エピローグを出力しない
_func2 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax           ; 配列 a の要素への代入
    movw    ax, sp
    call    !!_g
    br      !!__kernel_task_exit    ; タスク関数が常に出力する __kernel_task_exit 呼び出し
                                    ; エピローグを出力しない
_func3 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax           ; 配列 a の要素への代入
    movw    ax, sp
    call    !!_g
    call    !!_ext_tsk              ;ext_tsk 関数の呼び出し
    addw    sp, #0x06               ;#pragma rtos_task がないとエピローグを出力する
    ret
_func4 :
    subw    sp, #0x06
    onew    ax
    movw    [sp+0x00], ax           ; 配列 a の要素への代入
    movw    ax, sp
    call    !!_g
    movw    ax, [sp+0x00]
    or      a, x
    bnz     $.BB@LABEL@4_2
.BB@LABEL@4_1:                      ;return
    addw    sp, #0x06               ;#pragma rtos_task がないとエピローグを出力する
    ret
.BB@LABEL@4_2:                      ;bb3
    call    !!_ext_tsk
    br      $.BB@LABEL@4_1
```

関数のインライン展開（#pragma inline/#pragma noinline）

インライン関数であることをコンパイラに示唆します。

［機能］

- #pragma inline は，インライン展開する関数を宣言します。
- #pragma noinline は，-Oinline_level オプション使用時に特定関数のインライン展開を抑止する関数を宣言します。
- 同一翻訳単位内で，同じ関数に対して #pragma inline と #pragma noinline を同時に指定した場合はエラーとなります。
- #pragma inline 指令は，関数定義と同一翻訳単位内で，かつ関数定義より前に記述しなければなりません。
- #pragma inline の機能は，キーワード __inline と同一です。インライン展開の機能などについては「インライン関数指定（__inline）」を参照してください。

［方法］

- 対象関数より前で #pragma inline/#pragma noinline 宣言を行います。

```
#pragma inline      [(] 関数名 [,...][)]
#pragma noinline    [(] 関数名 [,...][)]
```

［使用例］

```
extern int      gi;

#pragma inline  i_func

static int i_func(int i)
{
        return ++i;
}

void func(int j)
{
        gi = i_func(j);
}
```

アセンブラ命令の記述（#pragma inline_asm）

アセンブリ記述関数をインライン展開します。

［機能］

- #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。

［方法］

- 対象関数より前で #pragma inline_asm 宣言を行います。

```
#pragma inline_asm  [(]関数名 [,...][)]
```

［制限］

- #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が生成されます。
- コンパイラは #pragma inline_asm を指定した関数内に書いた文字列を，そのままアセンブラに渡します。
- 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を #define でマクロ定義する場合は注意してください。iodefine.h をインクルードする場合は，アセンブリ記述関数を iodefine.h のインクルードより前に記述してください。
- アセンブラ制御命令は記述できません。また疑似命令については，以下に記載した命令のみを記述できます。下記以外の命令を記述した場合はエラーとなります。
  - データの定義，領域確保疑似命令（.DB/.DB2/.DB4/.DB8/.DS）
  - マクロ疑似命令（.MACRO/.IRP/.REPT/.LOCAL/.ENDM）
  - 外部定義疑似命令（.PUBLIC）【V1.04 以降】
- .PUBLIC 疑似命令には，アセンブリ記述関数内で定義したラベル以外を記述できません。記述した場合はエラーとなります。

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ます。この場合は，次のいずれかの方法で対処してください。

- アセンブリ記述のローカル・ラベルを使用してください。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。
  ローカル・ラベルについては，「.LOCAL」を参照してください。
- 外部ラベルは 1 箇所のみに展開されるように記述してください。
  - アセンブリ記述関数を同じソースファイル内から呼び出す場合は，アセンブリ記述関数定義に static を指定し，1 箇所のみからアセンブリ記述関数を呼び出してください。また，アセンブリ記述関数のアドレスを取得しないでください。
  - アセンブリ記述関数を同じソースファイル内から呼び出さない場合は，アセンブリ記述関数を外部関数としてください。

［使用例］

C ソースを以下に示します。

```
#pragma inline_asm func
void func(int x)
{
    movw !_a, ax
}

#pragma inline_asm func1
static void __near func1(void)  // 外部ラベル定義を含む inline_asm 指定関数を同一ファイルから
{                               // 呼び出す場合は，static __near 関数とする。
    .PUBLIC _label1
    incw ax
_label1:
    decw ax
}

#pragma inline_asm func2
void func2(void)                // 外部ラベル定義を含む inline_asm 指定関数を同一ファイルから
{                               // 呼び出さない場合は，外部関数とする。
    .PUBLIC _label2
    decw ax
_label2:
    incw ax
}

void main(void){
    func(3);
    func1();                    // 外部ラベル定義を含む inline_asm 指定関数を呼び出す。
}
```

コンパイラの出力アセンブリ・ソースは，以下のようになります。

```
    .SECTION .textf,TEXTF
_func:
    .STACK _func = 4
    ._line_top inline_asm
    movw !_a, ax
    ._line_end inline_asm
    ret
_func2:
    .STACK _func2 = 4
    ._line_top inline_asm
    .PUBLIC _label2
    decw ax
_label2:
    incw ax
    ._line_end inline_asm
    ret
_main:
    .STACK _main = 4
    movw ax, #0x0003
    ._line_top inline_asm   ; func の展開コード
    movw !_a, ax            ;
    ._line_end inline_asm   ;
    ._line_top inline_asm   ; func1 の展開コード
    .PUBLIC _label1         ;
    incw ax                 ;
_label1:                    ;
    decw ax                 ;
    ._line_end inline_asm   ;
    ret
```

絶対番地配置指定（#pragma address）

絶対番地に配置する変数を定義したいモジュール中で #pragma address 宣言を行うことにより，任意のアドレスに変数を配置することができます。

［機能］

- 指定した変数を指定したアドレスに割り付けます。

［効果］

- 任意のアドレスに変数を配置することができます。

［方法］

- 絶対番地に配置する変数を定義したいモジュール中で，#pragma address 宣言を行います。

```
#pragma address [(] 変数名 = 絶対アドレス [,...][)]
```

注　　*絶対アドレス*：実効アドレス（C 言語表記の 2，8，10 または 16 進数定数で記述）

［制限］

- #pragma address 指定は，変数の宣言前にしてください。変数宣言後の #pragm aaddress 指定は作用しません（警告も出力されません）。
- 変数以外を指定した場合はエラーとなります。
- const 修飾した変数に対して #pragma address を指定する場合はエラーとなります。【V1.04 以前】

```
#pragma address i=0xf2000
const int       i = 0;          // エラー
```

- 初期値付きで宣言した変数に対して #pragma address を指定する場合はエラーとなります。【V1.04 以前】
- 初期値付きで宣言した const 修飾型でない変数に対して #pragma address を指定する場合はエラーとなります。【V1.05 以降】

```
#pragma address i=0xf2000
int     i = 0;                  // エラー
```

- #pragma address を同一の変数に対して複数回指定した場合はエラーとなります。

```
#pragma address i=0xf2000
#pragma address i=0xf2000       // エラー
int     i;
```

- 異なる変数に対して同一アドレスを指定した場合，もしくは変数のアドレスが重なった場合はエラーとなります。
- 暗黙，明示問わずに near と指定した変数に対して #pragma address 宣言する場合，指定した絶対アドレスが 0x0F0000 ～ 0x0FFFFF の範囲に無い場合はコンパイル・エラーとなります。指定した絶対アドレスが SFR 領域である場合はリンク・エラーとなります。【V1.04 以前】
- 暗黙，明示問わずに near と指定した const 修飾型でない変数に対して #pragma address 宣言する場合，指定した絶対アドレスが 0x0F0000 ～ 0x0FFFFF の範囲に無い場合はコンパイル・エラーとなります。指定した絶対アドレスが SFR 領域である場合はリンク・エラーとなります。【V1.05 以降】

```
#pragma address n_i1=0xF0000
char    __near n_i1;            // コンパイル可
#pragma address n_i2=0xEFFFF
char    __near n_i2;            // エラー
#pragma address n_i3=0xEFFFF
char    n_i3;                   // エラー
                                //-memory_model=small, medium ともに bss は near であるため
#pragma address f_i=0xEFFFF
char    __far f_i;              // コンパイル可
```

- 暗黙，明示問わずに，near と指定した const 変数に対して指定したアドレスがミラー元領域にない場合，リンク・エラーとなります。【V1.05 以降】

```
#pragma  address  i=0xf3000
const    int  i = 0;          // エラー
```

［使用例］

C ソースを以下に示します。

```
#pragma address (io=0x0ffe00)

int     io;                     //io は 0x0ffe00 番地に配置
func(void){
        io = 0;
}
```

アセンブリ・ソースにおける変数の宣言とセクション割り付けは以下となります。

```
        .PUBLIC         _io

        .SECTION        .bss, BSS
        .ORG            0xFFE00
_io:
        .DS             2
```

関数中では以下のコードが出力されます。

```
clrw    ax
movw    !LOWW(_io), ax
```

［備考］

- #pragma address 指定しても，変数に volatile 属性を自動付加することはしません。
  オプション -volatile を使うことで，#pragma address を含む全ての静的変数に volatile 属性を付加することができます。
  個別に volatile を付加する場合は，変数宣言に volatile を付けてください。
- 変数を特定の番地に割り当てるにあたっては，変数の整列条件を配慮してください。また，64K バイト境界をまたいで配置しないようにしてください。

saddr 領域利用（#pragma saddr）

saddr 領域に割り当てる変数であることをコンパイラに示唆します。

［機能］

- 初期値あり変数はセクション .sdata へ配置します。
- 初期値なし変数はセクション .sbss へ配置します。
- アドレス参照は，必ず near ポインタを返します。
- #pragma saddr を指定した外部変数，および関数内 static 変数は，saddr 領域に割り当てます。
- __far キーワードを付加した変数であっても，警告なしで，__near 指定したものとして扱います。

［効果］

- saddr 領域に対する命令は通常メモリに対する命令よりも短く，オブジェクト・コードが小さくなり，実行速度が向上します。

［方法］

- 変数の最初の宣言より前で #pragma saddr を宣言します。

```
#pragma saddr [(] 変数名 [,...][)]
```

［制限］

- 同一変数に対し複数の宣言があり，2 度目以降の宣言に作用する位置に #pragma saddr を記述した場合，動作を保証しません。
- 他の #pragma を指定すると，コンパイル・エラーとなります。

［使用例］

```
#pragma saddr saddr_var
extern int saddr_var;

void func(void)
{
        saddr_var = 0;
}
```

［備考］

- キーワード __saddr と #pragma saddr の違いについて
  - __saddr キーワードは，__near/__far キーワードとの混在を許さずコンパイル・エラーとなります。
  - #pragma saddr は，__near/__far キーワードを付加した変数でも警告なしで __saddr を指定したものとして扱います。

callt 関数（#pragma callt）

callt 関数であることをコンパイラに示唆します。

［機能］

- #pragma callt を指定した関数を，callt 命令で呼び出します。
  callt 命令は，callt 命令テーブルと呼ばれる領域（0x80 ～ 0xBF）に「関数定義の先頭アドレス」が格納された関数を，直接関数を呼ぶよりも短いコードで呼ぶことを可能にします。
- callt 関数の呼び出しには，関数名の先頭に ”@_” を付加したラベル名を使用し，callt 命令で呼び出します。
  関数末尾で callt 関数を呼び出す場合，callt 命令での呼び出しにならない場合があります。
- 呼ばれる関数は，C ソース中において，通常の関数と同様に扱います。
- __near 指定となり，アドレス参照は必ず near ポインタを返します。
- callt 関数は再配置属性 TEXT のセクションへ配置します。
- callt 命令テーブルはセクション .callt0 に配置します。
- __far キーワードを付加した関数であっても，警告なしで，__near を指定したものとして扱います。
- 同一関数に対して #pragma callt，#pragma near，#pragma far が重複した場合は，#pragma callt > #pragma near > #pragma far の優先度で #pragma が有効になります。

［効果］

- 2 バイト・コール命令での関数呼び出しとなるため，オブジェクト・コードのサイズが小さくなります。

［方法］

- 関数の最初の宣言より前で #pragma callt を宣言します。

```
#pragma callt [()] 関数名 [,...][()]
```

［制限］

- 同一関数に対し複数の宣言があり，2 度目以降の宣言に作用する位置に #pragma callt を記述した場合，動作を保証しません。#pragma callt は宣言より前に記述してください。
- 同一関数に対し #pragma near，#pragma far 以外の #pragma を指定すると，コンパイル・エラーとなります。
- 本 #pragma 指令の対象となる関数の宣言に __inline，および __callt を指定すると，コンパイル・エラーとなります。

［使用例］

```
#pragma callt callt_func1
extern void callt_func1(void);

void func1(void)
{
        callt_func1();
          :
}

#pragma callt callt_func2
extern void __far callt_func2(void);  // #pragma callt の影響で，警告なしで __near になります

void func2(void)
{
        callt_func2();
          :
}
```

［備考］

- キーワード __callt と #pragma callt の違いについて
  - __callt キーワードは，__far キーワードとの混在を許さずコンパイル・エラーとなります。
  - #pragma callt は，__far キーワードを付加した変数でも警告なしで __callt を指定したものとして扱います。

near/far 関数（#pragma near/#pragma far） 【V1.05 以降】

__near/__far を指定した関数であることをコンパイラに示唆します。

[機能]

- #pragma near を指定した関数を，__near 指定の関数として呼び出します。
  #pragma near を指定した関数のアドレス参照は必ず near ポインタを返し，関数は再配置属性 TEXT のセクションへ配置します。
  __near についてはメモリ配置領域指定（__near/__far）を参照してください。
- #pragma far を指定した関数を，__far 指定の関数として呼び出します。
  #pragma far を指定した関数のアドレス参照は必ず far ポインタを返し，関数は再配置属性 TEXTF のセクションに配置します。
  __far についてはメモリ配置領域指定（__near/__far）を参照してください。
- #pragma near を指定した関数は，__callt，__far キーワードを付加した関数であっても，警告なしでキーワードを無視します。
  callt 関数としたい場合は #pragma near ではなく #pragma callt を指定してください。
- #pragma far を指定した関数は，__callt，__near キーワードを付加した関数であっても，警告なしでキーワードを無視します。
- 同一関数に対して #pragma callt，#pragma near，#pragma far が重複した場合は，#pragma callt > #pragma near > #pragma far の優先度で #pragma が有効になります。

[方法]

- 関数の最初の宣言より前で #pragma near/#pragma far を宣言します。

```
#pragma near [() 関数名 [,...][)]
#pragma far [() 関数名 [,...][)]
```

[制限]

- 同一関数に対し複数の宣言があり，2 度目以降の宣言に作用する位置に #pragma near/#pragma far を記述した場合，動作を保証しません。#pragma near/#pragma far は宣言より前に記述してください。
- 同一関数に対し #pragma callt，#pragma near，#pragma far 以外の #pragma を指定すると，コンパイル・エラーとなります。
- 本 #pragma 指令の対象となる関数の宣言に __inline を指定すると，コンパイル・エラーとなります。

[使用例]

```
#pragma near func1,func3
#pragma far  func2,func3          // func3 は #pragma near が優先されます

extern void func1(void);          // #pragma near の影響で，警告なしで __near になります
extern void __near func2(void);   // #pragma far の影響で，警告無しで __far になります
extern void __callt func3(void); // #pragma near の影響で，警告なしで __near になります
                                  // 警告なしで __callt は無効になります

void  main(void)
{
    func1( );
    func2( );
    func3( );
       :
}
```

［備考］

- キーワード __near / __far と #pragma near/#pragma far の違いについて
  - #pragma near/#pragma far は複数の関数に対して同時に指定できます。
  - __far キーワードは __callt キーワードや __near キーワードとの混在を許さずコンパイル・エラーとなります。
  - #pragma near/#pragma far は警告なしで __callt/__near/__far キーワードを無効とします。

構造体のパッキング（#pragma pack/#pragma unpack） 【V1.05 以降】

構造体のパッキングを指定します。

［機能］

- #pragma pack 指定位置以降で宣言された構造体に対し，構造体をパッキングします。構造体メンバのアライメントを 1 にします。
- #pragma unpack 指定位置以降で宣言された構造体に対し，構造体をパッキングしません。
- オプション -pack と #pragma unpack を同時に指定した場合は，#pragma unpack を優先します。

［方法］

- 構造体の宣言より前で #pragma pack/#pragma unpack を宣言します。

```
#pragma pack
#pragma unpack
```

［使用例］

```
#pragma pack
struct s1 {
  char a;
  int b;    // アライメントを 1 にします
} st1;
#pragma unpack
struct s2 {
  char a;
  int b;    // アライメントを 1 にしません
} st2;
```

［制限］

- 同一の構造体に対し，パッキング指定の異なる C ソース・ファイルが混在した場合，動作を保証しません。
- #pragma pack によって整列条件が 2 バイトから 1 バイトになった構造体，共用体，または，それらのメンバのアドレスを，標準ライブラリ関数の実引数として渡した場合，動作を保証しません。
- #pragma pack によって整列条件が 2 バイトから 1 バイトになった構造体，または共用体メンバのアドレスを，整列条件が 2 バイトである型のポインタに渡して間接参照した場合，動作を保証しません。

スタック破壊検出コードの生成（#pragma stack_protector/#pragma no_stack_protector） 【Professional 版のみ】 【V1.02 以降】

関数の入口および出口にスタック破壊を検出するコードを生成します。

［機能］

- 関数の入口・出口にスタック破壊検出コードを生成します。スタック破壊検出コードとは次に示す 3 つの処理を実行するための命令列を指します。
  (1) 関数の入口で，ローカル変数領域の直前（0xFFFF 番地に向かう方向）に 2 バイトの領域を確保し，その領域に num で指定した値を格納します。
  (2) 関数の出口で，num を格納した 2 バイトの領域が書き換わっていないことをチェックします。
  (3) (2) で書き換わっている場合には，スタックが破壊されたとして __stack_chk_fail 関数を呼び出します。
- num には 0 から 65535 までの 10 進数または 16 進数の整数値を指定します。num の指定を省略した場合には，コンパイラが自動的に数値を指定します。
- __stack_chk_fail 関数はユーザが定義する必要があり，スタックの破壊検出時に実行する処理を記述します。
- __stack_chk_fail 関数を定義する際には，次の項目に注意してください。
  - 戻り値および引数の型を void 型とし，far 領域に配置してください。
  - static 関数にしないでください。
  - 通常の関数のように呼び出すことは禁止します。
  - __stack_chk_fail 関数は，オプション -stack_protector，-stack_protector_all と #pragma stack_protector に関わらずスタック破壊検出コードの生成の対象にはなりません。
  - 関数内では abort() を呼び出してプログラムを終了させるなど，呼び出し元であるスタックの破壊を検出した関数にリターンしないようにしてください。
- #pragma no_stack_protector が指定された関数は，-stack_protector オプション，-stack_protector_all オプションに関わらず，スタック破壊検出コードを生成しません。

［効果］

- スタックを上書きするか破損する悪質なコードまたはプログラム・エラーに対する保護を提供する。

［方法］

- 関数の最初の宣言より前で #pragma stack_protector/#pragma no_stack_protector を宣言します。

```
#pragma stack_protector [(] 関数名 [(num= 数値 )][, 関数名 [(num= 数値 )]][,...][)]
#pragma no_stack_protector [(] 関数名 [,...][)]
```

［制限］

- -stack_protector オプションまたは -stack_protector_all オプションと同時に使用した場合は，#pragma stack_protector/#pragma no_stack_protector 指定が有効になります。
- #pragma stack_protector で指定した関数が次のいずれかの関数として指定されている場合，エラーメッセージを出力します。
  #pragma inline，__inline キーワード，#pragma inline_asm，#pragma no_stack_protector，#pragma rtos_interrupt，#pragma rtos_task

2 進定数

C ソース上で 2 進定数を記述できます。

[機能]

- 整数定数が記述可能な位置に，2 進定数を記述することができます。

[効果]

- ビット列で定数を記述したい場合，8 進数や 16 進数などに置き換えずに直接記述することができ，可読性も良くなります。

[方法]

- 2 進定数の記述方法は，次のとおりです。

```
0b    2 進数字
0B    2 進数字
```

- 0b または 0B に続けて，0 または 1 の数字の並びを記述してください。
- 数字と数字の間に ”_” を 1 つ記述できます。
- 2 進定数の値は，2 を基数として計算されます。
- 2 進定数の型は，8 進定数，16 進定数と同じになります。

[使用例]

C ソースを以下に示します。

```
int i1, i2, i3;

i1 = 0b00101100;
i2 = 0b0001_1000;
i3 = 0B0_1_0_1_0_1_0_1_0_1_0_1_0_1_0_1;
```

コンパイラの出力オブジェクトは，以下の場合と同じとなります。

```
int i1, i2, i3;

i1 = 0x2c;
i2 = 0x18;
i3 = 0x5555;
```

[注意]

- -strict_std オプション指定時に 2 進定数を記述した場合はエラーとなります。【V1.06 以降】

4.2.7 組み込み関数

CC-RL には，下記の “ 組み込み関数 ” があります。組み込み関数は全て，関数定義内からのみ呼び出すことができます。
以下に，関数として記述できる命令を示します。

表 4.16 組み込み関数

組み込み関数	機能	形式
__DI	DI 命令を出力します。	void __DI(void);
__EI	EI 命令を出力します。	void __EI(void);
__halt	HALT 命令を出力します。	void __halt(void);
__stop	STOP 命令を出力します。	void __stop(void);
__brk	BRK 命令を出力します。	void __brk(void);
__nop	NOP 命令を出力します。	void __nop(void);
__rolb	x を 8 ビット幅で y 回左ローテートします。 ビット幅を越えるローテート数指定時の動作は未定義です。 ローテート数がビット幅を越える可能性がある場合は，ビット幅でマスクしてください。	unsigned char __rolb(unsigned char x, unsigned char y);
__rorb	x を 8 ビット幅で y 回右ローテートします。 ビット幅を越えるローテート数指定時の動作は未定義です。 ローテート数がビット幅を越える可能性がある場合は，ビット幅でマスクしてください。	unsigned char __rorb(unsigned char x, unsigned char y);
__rolw	x を 16 ビット幅で y 回左ローテートします。 ビット幅を越えるローテート数指定時の動作は未定義です。 ローテート数がビット幅を越える可能性がある場合は，ビット幅でマスクしてください。	unsigned int __rolw(unsigned int x, unsigned char y);
__rorw	x を 16 ビット幅で y 回右ローテートします。 ビット幅を越えるローテート数指定時の動作は未定義です。 ローテート数がビット幅を越える可能性がある場合は，ビット幅でマスクしてください。	unsigned int __rorw(unsigned int x, unsigned char y);
__mulu	(unsigned int)x * (unsigned int)y の符号なし乗算を行い，16 ビットの演算結果を返します。	unsigned int __mulu(unsigned char x, unsigned char y);
__mului	(unsigned long)x * (unsigned long)y の符号なし乗算を行い，32 ビットの演算結果を返します。	unsigned long __mului(unsigned int x, unsigned int y);
__mulsi	(signed long)x * (signed long)y の符号付き乗算を行い，32 ビットの演算結果を返します。	signed long __mulsi(signed int x, signed int y);
__mulul	(unsigned long long)x * (unsigned long long)y の符号なし乗算を行い，64 ビットの演算結果を返します。-lang=c かつ -strict_std オプション指定時は組み込み関数として定義されません。	unsigned long long __mulul(unsigned long x, unsigned long y);
__mulsl	(signed long long)x * (signed long long)y の符号付き乗算を行い，64 ビットの演算結果を返します。-lang=c かつ -strict_std オプション指定時は組み込み関数として定義されません。	signed long long __mulsl(signed long x, signed long y);

組み込み関数	機能	形式
__divui	x と y の符号なし除算を行い，16 ビットの演算結果を返します。 除数 y が 0 の場合は 0xFFFF を返します。	unsigned int __divui(unsigned int x, unsigned char y);
__divul	x と y の符号なし除算を行い，32 ビットの演算結果を返します。 除数 y が 0 の場合は 0xFFFFFFFF を返します。	unsigned long __divul(unsigned long x, unsigned int y);
__remui	x と y の符号なし剰余算を行い，8 ビットの演算結果を返します。 除数 y が 0 の場合は被除数 x の下位 8 ビットを返します。	unsigned char __remui(unsigned int x, unsigned char y);
__remul	x と y の符号なし剰余算を行い，16 ビットの演算結果を返します。 除数 y が 0 の場合は被除数 x の下位 16 ビットを返します。	unsigned int __remul(unsigned long x, unsigned int y);
__macui	(unsigned long) x * (unsigned long) y + z の符号なし積和演算を行い，32 ビットの演算結果を返します。	unsigned long __macui(unsigned int x, unsigned int y, unsigned long z);
__macsi	(signed long) x * (signed long) y + z の符号付き積和演算を行い，32 ビットの演算結果を返します。	signed long __macsi(signed int x, signed int y, signed long z);
__get_psw	PSW の内容を返します。	unsigned char __get_psw(void);
__set_psw	PSW に x を設定します。	void __set_psw(unsigned char x);
__set1	set1 命令を使用し，x が指すアドレスのビット y を 1 にセットします。 ビット y は 0 から 7 までの定数のみ指定可能とし，それ以外はコンパイルエラーとします。	void __set1(unsigned char __near *x, unsigned char y);
__clr1	clr1 命令を使用し，x が指すアドレスのビット y を 0 にクリアします。 ビット y は 0 から 7 までの定数のみ指定可能とし，それ以外はコンパイルエラーとします。	void __clr1(unsigned char __near *x, unsigned char y);
__not1	not1 命令 (saddr 領域なら xor 命令) を使用し，x が指すアドレスのビット y を反転します。 ビット y は 0 から 7 までの定数のみ指定可能とし，それ以外はコンパイルエラーとします。	void __not1(unsigned char __near *x, unsigned char y);

5. アセンブラ言語仕様

この章では，CC-RL がサポートするアセンブラ言語仕様について説明します。

5.1 ソースの記述方法

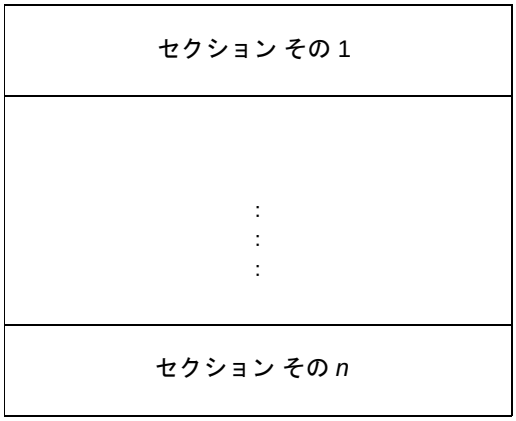
この節では，ソースの記述方法，式と演算子などについて説明します。

5.1.1 基本構成

ソース・プログラムの一般的な構成について説明します。

(1) セクション
ソース・プログラムは，セクションと呼ばれるブロックで構成されます。

ソース・プログラム＝セクションの集まり



セクションにはコード，データの 2 つのタイプがあります。
各セクションは独立したロケーション・カウンタを持ち，各ロケーション・カウンタは，リロケータブルなセクションの場合はその先頭からの相対アドレス値を保持し，アブソリュートなセクションの場合はメモリ空間内の絶対アドレス値を保持します。
各セクションは最適化リンカで任意のアドレスに配置できます。ただし，アブソリュートなセクションの場合はソース・プログラムで指定したアドレスから変更することはできません。

(2) モジュール
モジュールとは本アセンブラが一度に処理するソース・プログラムの単位であり，１つのモジュールは１つのアセンブリ・ソース・ファイルと対応しています。

5.1.2 記述方法

アセンブリ言語のソース・プログラムは，文で構成されます。
1 つの文は，1 行以内に「(1) 文字セット」で表される文字を使って記述します。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 ]
```

各欄は，空白，タブ，コロン（ : ），またはセミコロン（ ; ）で区切ります。1 行の最大文字数は 4294967294（=0xFFFFFFFE）（理論値）です。ただし，実際にはメモリ量により制限されます。
文の記述方法は自由記述形式で，シンボル欄，ニモニック欄，オペランド欄，コメント欄の順序が正しければ任意のカラムから記述することができます。ただし，1 つの文は 1 行以内で記述しなければなりません。

シンボル欄にシンボルを記述する場合は，コロン，1 つ以上の空白，またはタブで区切ります。ただし，コロンか空白かタブかはニモニックで記述する命令によります。また，コロンの前後には，任意の数の空白，またはタブを記述することができます。

オペランド欄の記述が必要な場合は，1 つ以上の空白，またはタブで区切ります。

コメント欄にコメントを記述する場合は，セミコロンで区切ります。セミコロンの前後には，任意の数の空白，またはタブを記述することができます。

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。
言語文字をさらに機能別に細分類すると下表になります。

表 5.1 言語文字の文字セット

細分類の総称名		文字
数字		0 1 2 3 4 5 6 7 8 9
英字	大文字	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	小文字	a b c d e f g h i j k l m n o p q r s t u v w x y z
英字相当文字		@ _（アンダースコア）.（ピリオド）
特殊文字	特殊 1	. , : ; * / + - ‘ < > ( ) $ = ! & # [ ] “ % << >> | ^ ? ~ △
	特殊 2	¥
	特殊 3	LF，CR LF，HT

- 英字（英字相当文字も英字に含める）と数字をまとめて英数字といいます。
- 予約語，または数値定数に記述した英小文字は，対応する英大文字として解釈されます。
- 英小文字をユーザ定義シンボルの中に使った場合，英大文字，英小文字の区別をして解釈されます。

特殊 1 文字の用途を以下に示します。
以下の使用方法以外で，文字データ・注釈（コメント）部以外のソース・プログラム中に現れたときはエラーとなります。

表 5.2 特殊 1 文字とその用途

文字	用途
.（ピリオド）	ビット位置指定子 疑似命令の開始記号
,（カンマ）	オペランドの区切り
:（コロン）	ラベルの区切り 拡張アドレス指定（”ES:”）
;（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 加算演算子
-（ハイフン）	負符号 減算演算子

文字	用途
'（シングル・クォーテーション）	文字定数の開始・終了記号
<	比較演算子 シフト演算子
>	比較演算子 シフト演算子
( )	演算順序の指定
$	制御命令の開始記号 相対アドレッシング開始記号 英字相当文字
=	比較演算子
!	比較演算子 絶対アドレッシングの開始
&	ビット論理演算子 論理演算子
#	イミディエート値の開始 コメントの開始（行頭の場合）
[ ]	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
%	剰余演算子
|	ビット論理演算子 論理演算子
^	ビット論理演算子
?	コンカティネート記号（マクロボディ内で使用）
~	ビット論理演算子
△（空白，またはタブ）	各欄の区切り記号

特殊 2 文字の用途を以下に示します。

表 5.3 特殊 2 文字とその用途

エスケープ・シーケンス	値（ASCII）	意味
¥0	0x00	null 文字
¥a	0x07	アラート（警告音）
¥b	0x08	バックスペース
¥f	0x0C	フォーム・フィード（改ページ）
¥n	0x0A	ニュー・ライン（改行）
¥r	0x0D	キャリッジ・リターン（復帰）
¥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・クォーテーション

エスケープ・シーケンス	値（ASCII）	意味
¥”	0x22	ダブル・クォーテーション
¥?	0x3F	疑問符
¥*ooo*	0 ～ 0377	3 桁まで（10 進表記で 0 ～ 255）の 8 進数（*o* は 8 進数字）
¥x*hh*	0x00 ～ 0xFF	2 桁まで（10 進表記で 0 ～ 255）の 16 進数（*h* は 16 進数字）

特殊 3 文字の用途を以下に示します。

<1> CR LF，LF
行の区切りを示す文字です。

特殊 3 文字	リスト出力
CR LF	0x0D0A
LF	0x0A

<2> HT
ソース・プログラム文のカラム位置を移動させる文字です。リストにはそのまま出力されます。

(b) 文字データ
文字データは，文字定数，文字列定数，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 すべての文字（マルチバイト文字を含みます。ただし，OS によってコードは異なります）が記述可能です。

- 英小文字と英大文字は区別して扱います。
- HT，CR LF，LF の扱いを下記に示します。

	オブジェクト出力	リスト出力
HT	0x09	0x09（タブはそのまま展開）
CR LF	0x0D0A	0x0D0A 注
LF	0x0A	0x0A 注

注 行の区切り文字であるため，文字データの一部とはみなされません。

(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) 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，次の 3 つがあります。

- 数値定数
- 文字定数
- 文字列定数

(a) 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”，または “0B” を記述 数値の最後に “b”，または “B” を記述	0b1101，0B1101 1101b，1101B
8 進数	数値の前に “0” を記述 数値の最後に “o”，または “O” を記述	074 074o，074O
10 進数	数値をそのまま記述	128
16 進数	数値の前に “0x”，または “0X” を記述 数値の最後に “h”，または “H” を記述	0xA6，0XA6 6Ah，6AH

先頭文字は数字でなければなりません。
例えば，10 進数の 10 を 16 進数で数値の最後に “H” を付けて表記する場合，先頭に “0” をつけて “0AH” と記述してください。“AH” と記述した場合はシンボルとみなされます。

注意 Prefix 表現（0xn…n など）と Suffix 表現（n…nh など）は，1 つのソース上での混在はできません。
オプション -base_number = (prefix | suffix) で表現形式を指定してください。

(b) 文字定数
文字定数は，1 つの文字をシングル・クォーテーション（’）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。
文字の長さは，1 とします。
値は，コード値を下位バイトから格納した 32 ビット値です。上位バイトが空いている場合は 0 で埋められます。

例

文字定数	評価値
'A'	0x00000041
' '（空白 1 個）	0x00000020

(c) 文字列定数
文字列定数は，文字そのものを表す文字の列で，「(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

文字列定数	評価値
"ab"	0x6162
"A"	0x41
" "（空白 1 個）	0x20
""	なし

(3) シンボル
シンボルを参照した場合，そのシンボルに対して定義した値を指定したものとして扱われます。
本アセンブラでのシンボルは，次の種類に分けられます。

- ネーム
シンボル定義疑似命令のシンボル欄に記述したシンボルです。このシンボルは値を持ちます。
値の範囲は -2147483648 ～ 2147483647（0x80000000 ～ 0x7FFFFFFF）です。

- ラベル
行の先頭から ':' までの部分に記述したシンボルです。このシンボルはアドレス値を持ちます。
アドレス値の範囲は 0 ～ 1048575（0x00000 ～ 0xFFFFF）です。

- 外部参照名
  あるモジュールで定義したシンボルを，ほかのモジュールで参照するために，外部参照名定義疑似命令のオペランド欄に記述したシンボルです。このシンボルのアドレス値はアセンブル時には 0 とし，リンク時に決定されます。
  また，参照しているモジュール内に定義がないシンボルも外部参照名とみなされます。
- セクション名
  セクション定義疑似命令のシンボル欄に記述したシンボルです。
  このシンボルは，値を持ちません。
- マクロ名
  マクロ定義疑似命令のシンボル欄に記述したシンボルです。マクロ参照時に使用されます。
  このシンボルは，値を持ちません。
- マクロ仮パラメータ名
  マクロ定義疑似命令のオペランド欄に記述したシンボルです。
  このシンボルは値を持ちません。

シンボルの定義時にビット位置指定子を用いて定義したシンボルを，ビット・シンボルと呼びます。
また，シンボルの参照時にビット位置指定を用いる参照を，シンボルのビット参照と呼びます。

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
シンボル欄に記述できるシンボルは，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
ラベル	ラベルの位置のアドレスを参照するために使用します。 なお，疑似命令に付加されたラベルは，その疑似命令の直前のセクションに含まれるとみなされます。	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。
ネーム	数値データやアドレスを割り付けて，それらの数値をシンボルとして参照するために使用します。	シンボル定義疑似命令のシンボル欄に記述します。 シンボル欄とニモニック欄は 1 個以上の空白，またはタブで区切ります。
セクション名	最適化リンカの入力情報として参照するために使用します。	セクション定義疑似命令のシンボル欄に定義します。 シンボル欄とニモニック欄は 1 個以上の空白，またはタブで区切ります。
マクロ名	ソース中で，マクロ参照時に使用します。	マクロ疑似命令のシンボル欄に記述します。 シンボル欄とニモニック欄は 1 個以上の空白，またはタブで区切ります。

シンボル欄に複数のシンボルを記述することはできません。また，1 行にいずれかのシンボルを 1 個しか定義できません。

(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。
  ピリオドを含むシンボルをビット操作命令のオペランドに記述する場合は，シンボルを 2 重引用符で囲んでください。
  例）set1 !"s.y.m".7
- 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，「5.6 予 約 語」を参照してください。
  また，デバイス・ファイル読み込み時には，SFR 略号，および拡張 SFR 略号をシンボルとして定義することもできません。SFR 略号，および拡張 SFR 略号の詳細は，デバイスのユーザーズ・マニュアルを参照してください。
- 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。
- 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。
  それ以外は空白，またはタブでニモニック欄と区切ります。

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    .EQU     0x10        ; “VAR01” はネーム
LAB01:   .DB2     0           ; “LAB01” はラベル
```

誤ったシンボルの例を以下に示します。

```
1ABC     .EQU     0x3         ; 先頭文字に数字は使用できません。
LAB      MOV      A, r10      ; “LAB” ラベルです。ニモニック欄とコロン（ : ）で区切ります。
FLAG:    .EQU     0x10        ; ネームにはコロン（ : ）が必要ありません。
```

シンボルのみで構成され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

アセンブラ生成シンボル（5.7 アセンブラ生成シンボルを参照）を記述する場合，多重定義でエラーとなる可能性があるので，アセンブラ生成シンボルは使用しないでください。
また，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する場合があるので注意しなければなりません。

(4) 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，またはタブで区切ります。
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      A, #1
```

誤った例を以下に示します。

```
MOVA, #1          ; 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     A, #1    ; ニモニック中に空白があります。
MOVE     A, #1    ; ニモニック欄に記述できない命令です。
```

(5) 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。なお，コンマの前後には任意の個数の空白，またはタブを記述することができます。

(6) コメント

コメント欄には，行頭のシャープ（#），または，行中のセミコロン（ ; ）のあとにコメント（注釈）を記述します。
コメント欄は，シャープ，または，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，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。

例

```
# これは コメント です
HERE:   MOV     A, #0x0F        ; これは コメント です
;
;       BEGIN LOOP HERE
;
```

5.1.3 式と演算子

式とは，シンボル[注1]，定数（数値定数[注2]，文字定数），前述の 2 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。

注 1. 式の要素として指定可能なシンボルは，ネーム，ラベル，外部参照名のみです。ただし，SIZEOF 演算子，および STARTOF 演算子に対してはセクション名を指定可能です。

注 2. デバイス・ファイル読み込み時には，式の中で SFR 略号，および拡張 SFR 略号を定数と同様に扱うことができます。

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。項のリロケーション属性により，使用可能な演算子が限られます。

演算子には「表 5.4 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 5.5 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。

表 5.4 演算子の種類

演算子の種類	演算子
算術演算子	+，-，*，/，%，+ 符号，- 符号
ビット論理演算子	~，&，|，^
比較演算子	==，!=，>，>=，<，<=
論理演算子	&&，||
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，MIRHW，MIRLW，SMRLW
特殊演算子	DATAPOS，BITPOS
セクション演算子	STARTOF，SIZEOF
その他の演算子	( )

上記の演算子は，単項演算子，特殊単項演算子，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，MIRHW，MIRLW，SMRLW，DATAPOS，BITPOS，STARTOF，SIZEOF
2 項演算子	+，-，*，/，%，&，|，^，==，!=，>，>=，<，<=，&&，||，>>，<<

表 5.5 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，~，HIGH，LOW，HIGHW，LOWW，MIRHW，MIRLW，SMRLW，DATAPOS，BITPOS，STARTOF，SIZEOF
	2	*，/，%，>>，<<
	3	+，-
	4	&，|，^
低い	5	==，!=，>，>=，<，<=
	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，32 ビットで行います。
  式中に記述した定数，式の評価途中，および評価結果が 32 ビットを越えた場合は，下位 32 ビットを有効とします。その場合エラーは出力されません。
  ただし，.DB8 疑似命令のオペランドに記述された式では，各々の項を 64 ビットとして扱います。
- 演算の各項は符号なし整数として扱われますが，以下の場合は符号付き整数として扱われます。
  乗算，除算，剰余算，論理シフトの第 2 項
- 除数がゼロの場合は，エラーとなります。
- 負の値は，2 の補数形式となります。
- リロケータブル項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 5.6 式評価の例

式	評価値
5 + 8 - 6 * 2 / 4	10
5 + (8 - 6) * 2 / 4	6
(5 + 8 - 6)* 2 / 4	3
2 * (0x0F - (0x0B & (0x0A | 0x0F)))	8
2 * 0x0F - 0x0B & 0x0A | 0x0F	0x0F
HIGH(-1)	0xFF
HIGH(0x0FFFF)	0xFF
2 + 4 * 5	22
(2 + 3) * 4	20
10/4	2
0 - 1	0xFFFFFFFF
-1 > 1	1（真）
EXT 注 + 1	0

注 EXT：外部参照名

5.1.4 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］

```
START:  BR      !!START + 6     ; (1)
```

(1) BR 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］

```
BACK:    BR       !!BACK - 6      ; (1)
```

(1) BR 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］

```
MOV     A, #2 * 3       ; (1)
```

(1)　MOV 命令実行により 6 を A レジスタにロードします。

```
/
```

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     A, #250 / 50    ; （1）
```

(1)　MOV 命令実行により 5 を A レジスタにロードします。

```
%
```

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     A, #256 % 50    ; (1)
```

(1)　MOV 命令実行により 6 を A レジスタにロードします。

+ 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。

- 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。

5.1.5 ビット論理演算子

ビット論理演算子には，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     A, #LOW(~3)     ; （1）
```

(1) “0x3” の論理否定をとります。
よって，“0xFC” を A レジスタにロードします。

NOT）	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     A, #0x6FA & 0x0F        ; (1)
```

(1) “0x6FA” と “0x0F” の論理積をとります。
よって，“0x0A” を A レジスタにロードします。

	0000	0000	0000	0000	0000	0110	1111	1010
AND）	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     A, #0x0A | 0b1101       ; (1)
```

(1) “0x0A” と “0b1101” の論理和をとります。
よって，“0x0F” を A レジスタにロードします。

	0000	0000	0000	0000	0000	0000	0000	1010
OR）	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］

```
MOV     A, #0x9A ^ 0x9D         ; (1)
```

(1) “0x9A” と “0x9D” の排他的論理和をとります。
よって，“0x07” を A レジスタにロードします。

	0000	0000	0000	0000	0000	0000	1001	1010
XOR）	0000	0000	0000	0000	0000	0000	1001	1101
	0000	0000	0000	0000	0000	0000	0000	0111

5.1.6 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 1（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 1（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 1（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 1（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 1（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 1（真），大きいときに 0（偽）を返します。

5.1.7 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
&&	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。

5.1.8 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。

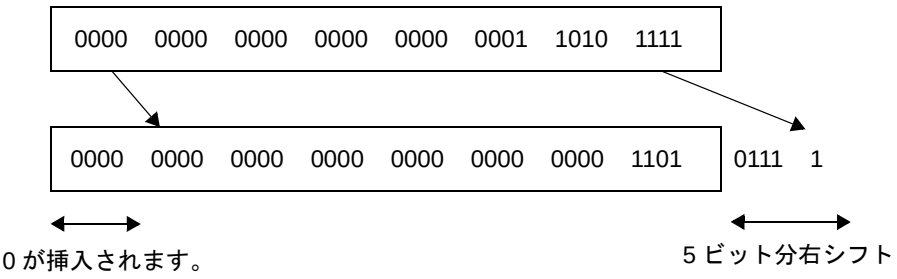
［機能］

第 1 項の値を第 2 項で示す値（ビット数）分だけ論理右シフトし，その値を返します。
上位ビットには，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］

```
MOVW    AX, #0x01AF >> 5        ; (1)
```

(1) “0x01AF” を 5 ビット分右シフトします。
よって，“0x000D” を AX に転送します。



```
<<
```

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。

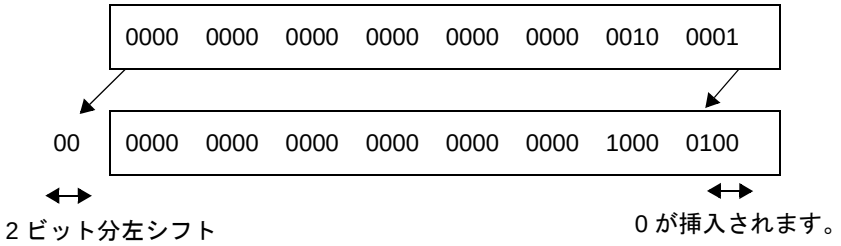
[機能]

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]

```
MOV     A, #0x21 << 2          ; (1)
```

(1) “0x21” を 2 ビット分左シフトします。
よって，“0x84” を A に転送します。



5.1.9 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	項の下位から 2 バイト目を求める
LOW	項の下位 8 ビットを求める

HIGH

項の下位から 2 バイト目を求めます。

［機能］

項の値の 32 ビット中（LSB をビット 0 として），ビット 8 ～ビット 15（下位から 2 バイト目）の値を返します。

［使用例］

```
MOV     A, #HIGH(0x1234)        ; (1)
```

(1) MOV 命令実行により，0x1234 の上位 8 ビット値 0x12 を A レジスタにロードします。

LOW

項の下位 8 ビットを求めます。

［機能］

項の値の 32 ビット中，下位 8 ビット値を返します。

［使用例］

```
MOV     A, #LOW(0x1234)         ; (1)
```

(1) MOV 命令実行により，0x1234 の下位 8 ビット値 0x34 を A レジスタにロードします。

5.1.10 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
MIRHW	項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの項の上位 16 ビットを求める
MIRLW	項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの項の下位 16 ビットを求める
SMRLW	シンボルのアドレスにミラー先までのオフセットを加算し，さらに整数値を加算した結果の 32 ビット値の下位 16 ビット値を求める

HIGHW

項の上位 16 ビットを求めます。

［機能］

項の値の 32 ビット中，上位 16 ビット値を返します。

［使用例］

```
MOVW    AX, #HIGHW(0x12345678)          ; (1)
```

(1)　MOVW 命令実行により，0x12345678 の上位 16 ビット値 0x1234 を AX レジスタにロードします。

LOWW

項の下位 16 ビットを求めます。

［機能］

項の値の 32 ビット中，下位 16 ビット値を返します。

［使用例］

```
MOVW    AX, #LOWW(0x12345678)          ; (1)
```

(1)　MOVW 命令実行により，0x12345678 の下位 16 ビット値 0x5678 を AX レジスタにロードします。

MIRHW

項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの項の上位 16 ビットを求めます。

［機能］

項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの 32 ビット中，上位 16 ビット値を返します。
項の値がミラー元領域の範囲外である場合は，アブソリュート項（5.1.14 演算の制限を参照してください）であれば HIGHW と同じ値を返します。リロケータブル項であればリンク時にエラーとなります。

［使用例］

```
MOVW    AX, #MIRHW(0x00001000)          ; (1)
```

(1) 演算対象となる式 0x00001000 がミラー元領域の範囲内にある場合，0x00001000 をミラー先アドレス（8 ビット CPU であれば 0x000F9000，16 ビット CPU であれば 0x000F1000）に変換し，上位 16 ビット値 0x000F を MOVW 命令実行により AX レジスタにロードします。

MIRLW

項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの項の下位 16 ビットを求めます。

［機能］

項の値がミラー元領域の範囲内にある場合は，ミラー先領域アドレスの 32 ビット中，下位 16 ビット値を返します。
項の値がミラー元領域の範囲外である場合は，アブソリュート項（5.1.14 演算の制限を参照してください）であれば LOWW と同じ値を返します。リロケータブル項であればリンク時にエラーとなります。

［使用例］

```
MOVW    AX, #MIRLW(0x00001000)          ; (1)
```

(1) 演算対象となる式 0x00001000 がミラー元領域の範囲内にある場合，0x00001000 をミラー先アドレス（8 ビット CPU であれば 0x000F9000，16 ビット CPU であれば 0x000F1000）に変換し，下位 16 ビット値（8 ビット CPU であれば 0x9000，16 ビット CPU であれば 0x1000）が，MOVW 命令実行により AX レジスタにロードします。

SMRLW

シンボルのアドレスにミラー先までのオフセットを加算し，さらに整数値を加算した結果の 32 ビット値の下位 16 ビット値を求めます。

[機能]

項がリロケータブルなシンボルに整数値を加算する式である場合，項の値ではなくシンボルのみに着目して，シンボルがミラー元領域の範囲内にある場合は，シンボルのアドレスにミラー先までのオフセットを加算し，さらに整数値を加算した結果の 32 ビット値の下位 16 ビット値を返します。
リロケータブルなシンボルがミラー元領域の範囲外にある場合は，リンク時にエラーとなります。
項がアブソリュート項である場合は LOWW と同じ値を，項がリロケータブルなシンボル参照のみの場合は MIRLW と同じ値を返します。

項として，アセンブラでの演算後に以下の形式となるもののみを許し，それ以外の場合はエラーとなります（A はアブソリュート・シンボル，R はリロケータブル・シンボル，C は整定数）。
MIRLW，MIRHW も同様です。

- SMRLW(C) ： LOWW と同等
- SMRLW(A) ： LOWW と同等
- SMRLW(R) ： R をミラー元か否かの判断対象とする
- SMRLW(R ＋ C) ： R をミラー元か否かの判断対象とする
- SMRLW(R － A ＋ C) ： R をミラー元か否かの判断対象とする

[使用例]

```
MOVW    AX, #SMRLW(GSYM + 0x1000)       ; (1)
```

GSYM は外部参照名です。

(1) リロケータブルなシンボル GSYM のアドレスがミラー元領域の範囲内にある場合，ミラー先アドレスに変換し，その結果に 0x1000 を加算した値の下位 16 ビット値を AX レジスタにロードします。

5.1.11 特殊演算子

特殊演算子には，次のものがあります。

演算子	概要
DATAPOS	ビット・シンボルの第 1 項を求める
BITPOS	ビット・シンボルの第 2 項を求める

DATAPOS

ビット・シンボルの第 1 項を求めます。

[機能]

ビット・シンボルの第 1 項を返します。

[使用例]

```
BITSYM  .EQU    0x0FE20.3
        MOVW    AX, #DATAPOS(BITSYM)    ; (1)
```

(1) MOVW 命令実行により 0xFE20 を AX レジスタへロードします。

[注意事項]

ビット項（「5.1.15 ビット位置指定子」を参照）はオペランド欄に記述できません。

BITPOS

ビット・シンボルの第 2 項を求めます。

[機能]

ビット・シンボルの第 2 項を返します。

[使用例]

```
BITSYM  .EQU    0x0FE20.3
        MOVW    AX, #BITPOS(BITSYM)     ; (1)
```

(1) MOVW 命令実行により 3 を AX レジスタへロードします。

[注意事項]

ビット項（「5.1.15 ビット位置指定子」を参照）はオペランド欄に記述できません。

5.1.12 セクション演算子

セクション演算子には，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]

4 バイト領域を確保し，デフォルト・セクション（.text）の先頭アドレスで初期化します。

```
.DB4            STARTOF(.text)
```

4 バイト領域を確保し，ユーザ定義セクション（user_text）の先頭アドレスで初期化します。

```
.DB4            STARTOF(user_text)
```

SIZEOF 演算子と組み合わせて指定する場合

```
.DB4            STARTOF(.data) + SIZEOF(.data)
```

[注意事項]

- 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。下記はエラーとなります。

```
.DB4            STARTOF(.data) +2
```

- アブソリュートなセクションは，“_AT” ＋指定アドレスを付けたセクション名（「.SECTION」，「.CSEG」，「.DSEG」，「.ORG」を参照）を記述してください。

```
.SECTION        EX, DATA_AT     0xF2000
.DB4            STARTOF(EX_ATF2000)
```

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]

4 バイト領域を確保し，デフォルト・セクション（.text）のサイズで初期化します。

```
.DB4    SIZEOF(.text)
```

4 バイト領域を確保し，ユーザ定義セクション（user_text）のサイズで初期化します。

```
.DB4            SIZEOF(user_text)
```

STARTOF 演算子と組み合わせて指定する場合

```
.DB4            STARTOF(.data) + SIZEOF(.data)
```

アブソリュートなセクションの場合，セクション名には EX_ATF2000 を指定します。

```
.SECTION        EX, DATA_AT     0xF2000
.DB4            SIZEOF(EX_ATF2000)
```

[注意事項]

- 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。
- アブソリュートなセクションは，“_AT” ＋指定アドレスを付けたセクション名（「.SECTION」，「.CSEG」，「.DSEG」，「.ORG」を参照）を記述してください。

5.1.13 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     A, #(4 + 3) * 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

(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 5.5 演算子の優先順位」を参照してください。

5.1.14 演算の制限

式の演算は，項を演算子で結びつけて行います。項として記述できるものには，定数，ネーム，ラベルがあり，各項はリロケーション属性を持ちます。

各項の持つリロケーション属性の種類により，その項に対して演算可能な演算子が限られます。したがって，式を記述する場合には，式を構成する各項のリロケーション属性に留意することが大切です。

(1) 演算とリロケーション属性

式を構成する各項は，リロケーション属性を持ちます。
各項をリロケーション属性により分類すると，アブソリュート項，リロケータブル項に分けられます。
次に，演算におけるリロケーション属性の種類とその性質，およびそれに該当する項を示します。

表 5.7 リロケーション属性の種類

種類	性質	該当項
アブソリュート項	アセンブル時に値，定数が決定する項	- 定数 - 定数を定義したネーム
リロケータブル項	アセンブル時には値が決定しない項	- ラベル - ラベルを定義したネーム - .EXTERN 疑似命令で定義したラベル - .EXTBIT 疑似命令で定義したネーム - モジュール内に定義がないシンボル

演算可能な演算子と項の組み合わせをリロケーション属性により分類すると，次のようになります。

表 5.8 リロケーション属性による項と演算子の組み合わせ

演算子の種類	項のリロケーション属性			
	X：ABS Y：ABS	X：ABS Y：REL	X：REL Y：ABS	X：REL Y：REL
+ X	A	A	R	R
- X	A	A	—	—
~ X	A	A	—	—
HIGH X	A	A	R 注1	R 注1
LOW X	A	A	R 注1	R 注1
HIGHW X	A	A	R 注1	R 注1
LOWW X	A	A	R 注1	R 注1
MIRHW X	A	A	R 注2	R 注2
MIRLW X	A	A	R 注2	R 注2
SMRLW X	A	A	R 注2	R 注2
DATAPOS X.Y	—	—	—	—
BITPOS X.Y	—	—	—	—
DATAPOS X	A	A	—	—
BITPOS X	A	A	—	—
X + Y	A	R	R	—
X - Y	A	—	R	R

演算子の種類	項のリロケーション属性			
	X：ABS Y：ABS	X：ABS Y：REL	X：REL Y：ABS	X：REL Y：REL
X * Y	A	—	—	—
X / Y	A	—	—	—
X % Y	A	—	—	—
X >> Y	A	—	—	—
X << Y	A	—	—	—
X & Y	A	—	—	—
X | Y	A	—	—	—
X ^ Y	A	—	—	—
X == Y	A	—	—	—
X != Y	A	—	—	—
X > Y	A	—	—	—
X >= Y	A	—	—	—
X < Y	A	—	—	—
X <= Y	A	—	—	—
X && Y	A	—	—	—
X || Y	A	—	—	—

ABS ： アブソリュート項
REL ： リロケータブル項
A ： 演算結果がアブソリュート項になります。
R ： 演算結果がリロケータブル項になります。
ー ： 演算不可

注 1. X が MIRHW，MIRLW，SMRLW，DATAPOS 演算を行ったリロケータブル項でない場合にかぎり，演算可能です。

注 2. X が HIGH，LOW，HIGHW，LOWW，MIRHW，MIRLW，SMRLW，DATAPOS 演算を行ったリロケータブル項ではない場合にかぎり，演算可能です。

(2) 演算子の入れ子
HIGH ／ HIGHW ／ LOW ／ LOWW の各演算子は入れ子を行う事が出来ます。

(3) 絶対式と相対式
式は「絶対式」と「相対式」に分けて扱います。

(a) 絶対式
定数値を示す式を “ 絶対式 ” と呼びます。絶対式は，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式として扱われます。

<1> 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。
ただし，シンボル参照時点において未定義なシンボル，および値が確定しないシンボルの場合は，定数式として扱われません。

例

```
SYM1    .EQU    0x10            ; シンボル SYM1 を定義
        MOV     A, #SYM1        ; 定義済みの SYM1 は定数式として扱う
```

<2> 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，アブソリュート項であるシンボル，すなわち，モジュール内で定数で定義されたネームで，かつ，その時点において未定義なシンボル，および値の確定しない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

```
        MOV     A, #SYM1        ; この時点で SYM1 は未定義シンボル
SYM1    .EQU    0x10            ; SYM1 を定義
```

(b) 相対式
特定のアドレスからのオフセット値[注 1] を示す式を “ 相対式 ” と呼びます。相対式は，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，相対式は，シンボル（ラベル，外部参照名）によって構成されます。
次に示した形式[注 2] が相対式として扱われます（“±” は “+” か “-” のどちらかになります）。

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. “- 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。

- シンボル
- シンボル ± 定数式
- シンボル - シンボル[注]
- シンボル - シンボル ± 定数式[注]

注 “-” の後のシンボルとしてラベルを記述することはできません。ただし，ラベル同士の減算は除きます。

いずれかのシンボルがリロケータブル項であれば相対式となります。式の例は次のようになります。

例

```
SIZE    .EQU    0x10
        MOV     A, #label1
        MOV     A, #label1 + 0x10
        MOV     A, #label2 - SIZE
        MOV     A, #label2 - SIZE + 0x10
```

5.1.15 ビット位置指定子

ビット位置指定子（．）を使用することにより，ビット・アクセスが可能になります。

(1) 記述形式

アドレス．ビット位置

(2) 機能
第 1 項にアドレスを指定するもの，第 2 項にビット位置を指定するものを指定します。これにより，ビット・アクセスが可能になります。

(3) 説明

- ビット位置指定子によって得られたものを，ビット値を持つビット項と呼びます。
- ビット項は，式の項として記述することはできません。
- ビット位置指定子には演算子との優先順位はなく，左辺を第 1 項，右辺を第 2 項と認識します。
- 第 1 項には，次の制限があります。
  - ビット・データを扱う命令（MOV1 など）で指定可能なオペランドが記述できます（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）。
  - 第 1 項に絶対式を記述する場合は，0x00000 ～ 0 x FFFFF の範囲でなければなりません。
  - 外部参照名を記述することができます。
- 第 2 項には，次の制限があります。
  - 絶対式の値は，0 ～ 7 の範囲です。範囲を越えた場合にはエラーとなります。
  - 外部参照名を記述することはできません。

(4) 演算とリロケーション属性
リロケーション属性における第 1 項と第 2 項の組み合わせを次に示します。

項の組み合わせ X：	ABS	ABS	REL	REL
項の組み合わせ Y：	ABS	REL	ABS	REL
X.Y	A	—	R	—

ABS ： アブソリュート項
REL ： リロケータブル項
A ： 演算結果がアブソリュート項になります。
R ： 演算結果がリロケータブル項になります。
— ： 演算不可

(5) 使用例

```
MOV1    CY, 0xFFE20.3
AND1    CY, A.5
CLR1    P1.2
SET1    1 + 0xFFE30.3   ;0xFFE31.3 に等しい（(1 + 0xFFE30) が第 1 項，3 が第 2 項）
SET1    0xFFE40.4 + 2   ;0xFFE40.6 に等しい（0xFFE40 が第 1 項，(4 + 2) が第 2 項）
```

5.1.16 オペランドの特性

オペランドを必要とする命令（インストラクション，および疑似命令）は，その種類により要求するオペランド値のサイズ，範囲などが異なります。

たとえば，“MOV r, #byte” というインストラクションの機能は，「レジスタ r に，byte で示される値を転送する」ものです。このとき，レジスタ r は 8 ビット長のレジスタであるため，転送されるデータ “byte” のサイズは，8 ビット以下でなければなりません。

もし，“MOV R0, 0x100” と記述した場合には，第 2 オペランド “0x100” のサイズが 8 ビット長を越えているため，アセンブル・エラーとなります。

このように，オペランドを記述する場合には，次のような注意が必要です。

- 値のサイズ，アドレス範囲がその命令のオペランドに適しているかどうか（数値やネーム，ラベル）

(1) オペランドの値のサイズとアドレス範囲
命令のオペランドとして記述可能な数値／ネーム／ラベルの値のサイズとアドレス範囲には条件があります。
インストラクションの場合は，各インストラクションのオペランドの表現形式により，疑似命令の場合には命令の種類により，記述可能なオペランドのサイズとアドレス範囲に条件があります。
これらの条件を次に示します。

表 5.9 インストラクションのオペランド値の範囲

オペランドの表現形式	値の範囲	
byte	8 ビット値 ： 0x00 ～ 0xFF	
word	word [B] word [C] word [BC]	- 数値定数の場合 0x0000 ～ 0xFFFF - ラベルの場合 0xF0000 ～ 0xFFFFF 注 1 ただし，ラベルがミラー元領域注 2 に属する場合はミラー先領域注 2 の値を 16 ビット値にマスクした値となる
	ES:word [B] ES:word [C] ES:word [BC]	- 数値定数の場合 0x0000 ～ 0xFFFF - ラベルの場合 0x00000 ～ 0xFFFFF ES の値はチェックしない
	上記以外	16 ビット値 ： 0x0000 ～ 0xFFFF
saddr	0xFFE20 ～ 0xFFF1F 注 3 ただし，saddr 領域はデバイスにより異なります	
saddrp	0xFFE20 ～ 0xFFF1F の偶数値注 3 ただし，saddr 領域はデバイスにより異なります	
sfr	0xFFF20 ～ 0xFFFFF ： 特殊機能レジスタ略号（SFR 略号注 4），および数値定数，シンボル	
sfrp	0xFFF20 ～ 0xFFFFE ： 特殊機能レジスタ略号（SFR 略号注 4），および数値定数，シンボル（偶数値のみ）	
addr5	0x00080 ～ 0x000BF（CALLT 命令テーブル領域／偶数値のみ）	

オペランドの表現形式	値の範囲	
addr16	!addr16 （BR，CALL 命令）	0x0000 ～ 0xFFFF （数値定数，ラベルともに，指定可能な範囲は同じ）
	!addr16 注 5 （BR，CALL 以外の命令）	- 数値定数 の場合注 6 0x0000 ～ 0xFFFF - ラベルの場合注 6 0xF0000 ～ 0xFFFFF 注 1 ただし，ラベルがミラー元領域注 2 に属する場合はミラー先領域注 2 の値を 16 ビット値にマスクした値となる
	ES:!addr16	- 数値定数の場合注 6 0x0000 ～ 0xFFFF - ラベルの場合注 6 0x00000 ～ 0xFFFFF ES の値はチェックしない
	!addr16.bit	- addr16 部分が数値定数の場合 0x0000 ～ 0xFFFF - addr16 部分，または addr16.bit がシンボルの場合 0xF0000 ～ 0xFFFFF 注 1 ただし，ラベルがミラー元領域注 2 に属する場合はミラー先領域注 2 の値を 16 ビット値にマスクした値となる
	ES:!addr16.bit	- addr16 部分が数値定数の場合 0x0000 ～ 0xFFFF - addr16 部分，または addr16.bit がシンボルの場合 0x00000 ～ 0xFFFFF ES の値はチェックしない
addr20	$addr20	0x00000 ～ 0xFFFFF，かつ分岐命令の次のアドレスから分岐先までが (-0x80) ～ (+0x7F) の範囲
	$!addr20	0x00000 ～ 0xFFFFFF ，かつ分岐命令の次のアドレスから分岐先までが (-0x8000) ～ (+0x7FFF) の範囲
	!!addr20	0x00000 ～ 0xFFFFF
bit	3 ビット値 ： 0 ～ 7	
RBn	n:2 ビット値 ： 0 ～ 3	

注 1. ミラー先領域，および内部 RAM 領域とし，各領域はデバイス・ファイルを参照します。デバイス・ファイルを参照しない場合の範囲は 0xF0000 ～ 0xFFFFF とします。

注 2. ミラー元領域のアドレス範囲は，デバイスによって異なります。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。

注 3. saddr 領域はデバイス・ファイルを参照します。デバイス・ファイルを参照しない場合の範囲は 0xFFE20 ～ 0xFFF1F とします。

注 4. SFR 略号はデバイス・ファイルを参照します。デバイス・ファイルを参照しない場合は SFR 略号の使用は不可とします。なお，SFR 略号のアドレス範囲は 0xFFF00 ～ 0xFFFFF ですが，0xFFF00 ～ 0xFFF1F のアドレス範囲は，SFR 略号で記述されていても saddr とみなします。

注 5. SFR 略号，および拡張 SFR（2ndSFR）略号をオペランドに記述する場合，「!addr16」として「!SFR」や「!2ndSFR」の記述が可能とし，!addr16 としてコードが生成されます。
拡張 SFR 略号のアドレス範囲もデバイス・ファイルを参照します。

注 6. 16 ビット命令（16 ビット・データ転送命令，16 ビット演算命令など）のオペランドの場合は偶数値のみです。

数値定数とラベルとで記述可能な範囲が異なる理由について，次に示します。
オペランドの word，addr16 に対してコードを生成する場合，出力可能な値は 0x0000 ～ 0xFFFF です。したがって，オペランドに数値定数を記述する場合には，それぞれこの範囲でチェックを行います。しかし，オペランドにラベルを記述する場合には，それぞれの数値の意味を鑑み，以下の範囲としています。

(a) word
ベースト・アドレッシングでは，実際のアクセス先は 0xF0000 ～ 0xFFFFF です。したがって，ラベルを記述する場合には，このアドレス範囲であることをチェックします。

(b) addr16
BR，CALL 以外の命令では，実際のアクセス先は 0xF0000 ～ 0xFFFFF です。したがって，ラベルを記述する場合には，このアドレス範囲であることをチェックします。

表 5.10 疑似命令のオペランド値の範囲

種類	疑似命令	値の範囲
セクション定義	.ORG	0x00000 ～ 0xFFFFF
	.OFFSET	0x00000 ～ 0xFFFFF
シンボル定義	.EQU	0x00000000 ～ 0xFFFFFFFF ただし，ビット・シンボルの場合は， アドレス値 ：0x00000 ～ 0xFFFFF ビット値 ：0 ～ 7 です。
	.SET	0x00000000 ～ 0xFFFFFFFF
データ定義・領域確保	.DB	初期値指定 ：0x00 ～ 0xFF
	.DB2	初期値指定 ：0x0000 ～ 0xFFFF
	.DB4	初期値指定 ：0x00000000 ～ 0xFFFFFFFF
	.DB8	初期値指定 ：0x00000000 00000000 ～ 0xFFFFFFFF FFFFFFFF
	.DS	サイズ指定 ：0x00000 ～ 0xFFFFF
	.ALIGN	整列条件 ：2 以上 2^{31} 未満の偶数

(2) 命令の要求するオペランドのサイズ
命令には，機械命令と疑似命令がありますが，オペランドとしてイミーディエト・データ，またはシンボルを要求する命令については，各命令により要求するオペランドのサイズが異なります。したがって，命令の要求するオペランドのサイズ以上のデータを記述すると，エラーとなります。
また，式の評価は計算結果，および途中も含めて 32 ビットで行うため，オーバフローした場合でも 32 ビット値として扱います。
ただし，オペランドにリロケータブルなシンボルを記述した場合は，アセンブラ内では値が決定されないので，最適化リンカにおいて値の決定と範囲のチェックが行われます。

(3) 命令の要求するシンボルの性質
オペランドとしてシンボルが記述可能な命令でも，各命令により，記述できるシンボルの性質 (アブソリュート，リロケータブル，外部参照) が異なります。
シンボルの参照方向には，後方参照と前方参照があります。

- 後方参照：オペランドとして参照するシンボルがそれ以前の行で定義されています。
- 前方参照：オペランドとして参照するシンボルがそれ以降の行で定義されています。

例を以下に示します。

```
        .CSEG
L1 :                    <-- 後方参照
        BR      !L1     <--
        BR      !L2     <-- 前方参照
L2 :                    <--
```

次に，機械語命令のオペランドとして記述可能なシンボルの性質を示します。

表 5.11 オペランドとして記述可能なシンボルの性質

	リロケーション属性	アブソリュート		リロケータブル[注 1]		SFR 略号[注 2]	
	参照パターン	後方	前方	後方	前方	SFR	2ndSFR[注 3]
記述形式	byte	○	○	○	○	－	－
	word	○	○	○	○	－	－
	saddr	○	○	○	○	○[注 4,5]	－
	saddrp	○	○	○	○	○[注 4,6]	－
	sfr	－	－	－	－	○[注 4,7]	－
	sfrp	－	－	－	－	○[注 4,8]	－
	addr20	○	○	○	○	○	○
	addr16	○	○	○	○	○[注 9]	○[注 9]
	addr5	○	○	○	○	－	－
	bit	○	－	－	－	－	－

前方 ： 前方参照
後方 ： 後方参照
○ ： 記述可能
－ ： エラー

注 1. リロケータブル・シンボルを記述した場合は，最適化リンカによって値の決定，および範囲チェックを 行います。

注 2. .EQU 疑似命令のオペランドに，sfr，sfrp（saddr と sfr がオーバーラップしていない領域の sfr）を指定し，定義されたシンボルは後方参照のみとし，前方参照は禁止します。

注 3. 拡張特殊機能レジスタ（2nd SFR：2nd Special Function Register）

注 4. オペランドの組み合わせに，saddr/saddrp を sfr/sfrp に入れ替えた組み合わせが存在する命令に対し，saddr 領域の SFR 略号を記述した場合は，saddr/saddrp としてコードが出力されます。

注 5. saddr 領域の 8 ビット SFR

注 6. saddr 領域の 16 ビット SFR

注 7. 8 ビット SFR

注 8. 16 ビット SFR

注 9. BR，CALL 以外の命令のオペランド !addr16 でのみ，!SFR，!2ndSFR，および SFR の指定が可能です。

表 5.12 疑似命令のオペランドとして記述可能なシンボルの性質

	リロケーション属性	アブソリュート		リロケータブル[注 1]	
	参照方向	後方	前方	後方	前方
疑似命令	.ORG	○[注 2]	−	−	−
	.OFFSET	○[注 2]	−	−	−
	.EQU	○[注 2]	−	−	−
	.SET	○[注 2]	−	−	−
	.DB	○	○	○	○
	.DB2	○	○	○	○
	.DB4	○	○	○	○
	.DB8	○[注 2]	−	−	−
	.DS	○[注 2]	−	−	−
	.ALIGN	○[注 2]	−	−	−

前方 : 前方参照
後方 : 後方参照
○ : 記述可能
− : 記述不可能

注 1. リロケータブル・シンボルを記述した場合は，最適化リンカによって値の決定，および範囲チェックを 行います。

注 2. 絶対式のみが記述可能です。

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 5.13　　疑似命令一覧

種類	疑似命令
セクション定義疑似命令	.SECTION，.CSEG，.DSEG，.BSEG，.ORG，.OFFSET
シンボル定義疑似命令	.EQU，.SET
データ定義，領域確保疑似命令	.DB，.DB2，.DB4，.DB8，.DS，.DBIT，.ALIGN
外部定義，外部参照疑似命令	.PUBLIC，.EXTERN，.EXTBIT
コンパイラ出力疑似命令	.LINE，.STACK，._LINE_TOP，._LINE_END，.VECTOR
マクロ疑似命令	.MACRO，.LOCAL，.REPT，.IRP，.EXITM，.EXITMA，.ENDM
分岐疑似命令	.Bcond

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

セクション定義疑似命令は，セクションの開始，および終了を指示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

```
.SECTION    SecA, TEXT
    :
.SECTION    SecB, DATA
    :
.SECTION    SecC, BSS
    :
```

再配置属性 SBSS と SBSS_BIT のセクション同士は，同じセクション名であっても良いです。再配置属性 BSS と BSS_BIT のセクション同士は，同じセクション名であっても良いです。
その他の再配置属性に関しては，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。また，再配置属性によってセクションに記述できる内容は異なります。詳細は「表 5.15 再配置属性」を参照してください。

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。
別々のモジュール（アセンブリ・ソース・ファイル）内に分割記述した場合は，最適化リンカが結合します。

セクションは開始アドレスを指定することが出来ます。開始アドレスが指定されたセクションはアブソリュート・セクションです。

セクション定義疑似命令には，次のものがあります。

表 5.14 セクション定義疑似命令

疑似命令	概要
.SECTION	アセンブラにセクションの開始を指示します
.CSEG	アセンブラにコード・セクションの開始を指示します
.DSEG	アセンブラにデータ・セクションの開始を指示します
.BSEG	アセンブラにビット・セクションの開始を指示します
.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

.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
[*ラベル* :]	.SECTION	*セクション名* , *再配置属性* [, ALIGN= *絶対値式*][,COMDAT= *シグネチャ名*]	[; *コメント*]

ALIGN は V1.10 以降で記述可能です。
COMDAT は V1.12 以降で記述可能です。

[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。

[詳細説明]

- プログラム中のひとつのまとまった機能を持つプログラム，およびデータを .SECTION 疑似命令により定義します。
  本疑似命令は，次にセクション定義疑似命令が記述されるまで有効です。
- 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，ラベル，またはオブジェクト・コード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。
- .SECTION 疑似命令は，オペランド欄の再配置属性に AT，DATA_AT，BSS_AT，BIT_AT を指定することにより，開始アドレスを指定することができます。また，.ORG 疑似命令により開始アドレスを指定することもできます。このときのセクション名は，オペランドに指定されたセクション名＋ "_AT" ＋指定アドレス（ただし，prefix（0x, 0X）および suffix（h，H）が付かない大文字の 16 進表記）となります。
- 次に，再配置属性を示します。
  下記以外の再配置属性が指定された場合はエラーとなります。

表 5.15 再配置属性

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注1]
CALLT0	CALLT0	.callt0	セクションをコード・フラッシュ領域[注2] 内の 0x00080 ～ 0x000BF 番地内で先頭が偶数番地になるように配置します。	2
TEXT	TEXT	.text	セクションをコード・フラッシュ領域[注2] 内の 0x000C0 ～ 0x0FFFF 番地内に配置します。	1
TEXTF	TEXTF	.textf	セクションをコード・フラッシュ領域[注2] 内の 0x000C0 ～ 0xEFFFF 番地内に配置します。	1
TEXTF_UNIT64KP	TEXTF_UNIT64KP	.textf_unit64kp	セクションを先頭が偶数番地になるように，64KB-1 境界[注3] にまたがらないように配置します。	2[注4]
CONST	CONST	.const	セクションをミラー元領域[注2] に，先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注1]
CONSTF	CONSTF	.constf	セクションをコード・フラッシュ領域[注2]内で先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。	2
SDATA	SDATA	.sdata	初期値を持つデータ用のセクションを saddr 領域[注2]内で先頭が偶数番地になるように配置します。	2
SBSS	SBSS	.sbss	初期値を持たないデータ用のセクションを saddr 領域[注2]内で先頭が偶数番地になるように配置します。[注8]	2
SBSS_BIT	SBSS_BIT	.sbss_bit	初期値を持たないビット用のセクションを saddr 領域[注2]内で先頭が偶数番地になるように配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を SBSS として扱います。[注8][注9]	2
DATA	DATA	.data	初期値を持つデータ用のセクションを RAM 領域[注2]内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。	2
BSS	BSS	.bss	初期値を持たないデータ用のセクションを RAM 領域[注2]内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。[注8]	2
BSS_BIT	BSS_BIT	.bss_bit	初期値を持たないビット用のセクションを RAM 領域[注2]内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を BSS として扱います。[注8][注10]	2
DATAF	DATAF	.dataf	初期値を持つデータ用のセクションを先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。	2
BSSF	BSSF	.bssf	初期値を持たないデータ用のセクションを先頭が偶数番地になるように，64KB - 1 境界[注3]にまたがらないように配置します。	2
AT △アドレス	AT 絶対式[注5]	なし	セクションを指定番地に配置します。	1（固定）
DATA_AT △アドレス	DATA_AT 絶対式[注5]	なし	初期値を持つデータ用のセクションを指定番地に配置します。	1（固定）
BSS_AT △アドレス	BSS_AT 絶対式[注5]	なし	初期値を持たないデータ用のセクションを指定番地に配置します。[注8]	1（固定）
BIT_AT △アドレス	BIT_AT 絶対式[注5]	なし	初期値を持たないビット用のセクションを指定番地に配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を BSS_AT として扱います。[注8][注11]	1（固定）
OPT_BYTE	OPT_BYTE	.option_byte[注6]	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用の属性[注7]です。	1（固定）
SECUR_ID	SECUR_ID	.security_id[注6]	セキュリティ ID 指定専用の属性[注7]です。 機械語命令は記述できません。	1（固定）

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注 1]
FLASH_SECUR_ID	FLASH_SECUR_ID	.flash_security_id[注 6]	フラッシュ・プログラマ・セキュリティ ID 指定専用の属性[注 7] です。 機械語命令は記述できません。	1（固定）

注 1. 整列条件は .ALIGN 疑似命令にて変更することができます。

注 2. コード・フラッシュ領域，ミラー領域，RAM 領域，saddr 領域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。ただし，RAM 領域については，アドレス範囲が 0xF0000 ～ 0xFFFFF である内部 RAM のみサポートします。

注 3. 配置の境界制限について，デフォルトの設定を 64KB - 1 境界とします。

注 4. 16 ビット・データへのアクセスを保証するため，整列条件を「2」とします。

注 5. 絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。

注 6. 特別なセクションのため，セクション名の変更を禁止とし，セクション名固定とします。

注 7. オプション・バイト，およびオンチップ・デバッグ，セキュリティ ID の配置先アドレス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。

注 8. 再配置属性 SBSS と SBSS_BIT，再配置属性 BSS と BSS_BIT，再配置属性 BSS_AT と BIT_AT の同名セクションは，アセンブラ内部で連続した 1 つのセクションとして処理します。

注 9. セクションを再配置属性 SBSS としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 SBSS として配置します。

注 10. セクションを再配置属性 BSS としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 BSS として配置します。

注 11. セクションを再配置属性 BSS_AT としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 BSS_AT として配置します。

- セクション名の指定は省略できません。
- 再配置属性の指定は省略できません。
- 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（@，_，.）
- ALIGN パラメータを指定すると，整列条件のデフォルト値を変更することができます。ALIGN パラメータに指定した値より大きな値を .ALIGN 疑似命令に指定した場合，エラーになります。【V1.10.00 以降】
- COMDAT パラメータを指定すると，リンク時に同じシグネチャを持つ同名セクションの中から，１つのみを選択してリンクします。【V1.12.00 以降】

［使用例］

TEXT 属性のセクション “.text” を定義

```
.SECTION        .text , TEXT
NOP
```

DATA 属性のセクション “data” を定義

```
.SECTION        data, DATA
.DB2            0x1
```

DATA_AT 属性のセクション ”EX” を 0xf2000 番地指定で定義
セクション名は “EX_ATF2000” となります。

```
.SECTION        EX, DATA_AT     0xf2000
.DS             4
```

.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。

[詳細説明]

- プログラム中で，ひとつのまとまった機能を持つ部分を .CSEG 疑似命令により定義します。
  本疑似命令は，次にセクション定義疑似命令が記述されるまで有効です。
- 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，ラベル，またはオブジェクト・コード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。
- .CSEG 疑似命令は，オペランド欄に AT を指定することにより開始アドレスを指定することができます。
  また，.ORG 疑似命令により開始アドレスを指定することもできます。
  このときのセクション名は，指定されたセクション名＋ “_AT” ＋指定アドレス（ただし，prefix（0x，0X）および suffix（h，H）が付かない大文字の 16 進表記）となります。
- 次に，.CSEG の指定可能な再配置属性を示します。

表 5.16 .CSEG の再配置属性

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注1]
CALLT0	CALLT0	.callt0	セクションをコード・フラッシュ領域[注2] 内の 0x00080 ～ 0x000BF 番地内で先頭が偶数番地になるように配置します。	2
TEXT	TEXT	.text	セクションをコード・フラッシュ領域[注2] 内の 0x000C0 ～ 0x0FFFF 番地内に配置します。	1
TEXTF	TEXTF	.textf	セクションをコード・フラッシュ領域[注2] 内の 0x000C0 ～ 0xEFFFF 番地内に配置します。	1
TEXTF_UNIT64KP	TEXTF_UNIT64KP	.textf_unit64kp	セクションを先頭が偶数番地になるように，64KB-1 境界[注3] にまたがらないように配置します。	2[注4]
CONST	CONST	.const	セクションをミラー元領域[注2] に，先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
CONSTF	CONSTF	.constf	セクションをコード・フラッシュ領域[注2] 内で先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
AT △アドレス	AT 絶対式[注5]	なし	セクションを指定番地に配置します。	1（固定）

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注 1]
OPT_BYTE	OPT_BYTE	.option_byte[注 6]	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用の属性[注 7] です。	1（固定）
SECUR_ID	SECUR_ID	.security_id[注 6]	セキュリティ ID 指定専用の属性[注 7] です。 機械語命令は記述できません。	1（固定）
FLASH_SECUR_ID	FLASH_SECUR_ID	.flash_security_id[注 6]	フラッシュ・プログラマ・セキュリティ ID 指定専用の属性[注 7] です。 機械語命令は記述できません。	1（固定）

注 1. 整列条件は .ALIGN 疑似命令にて変更することができます。

注 2. コード・フラッシュ領域，ミラー領域，RAM 領域，saddr 領域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。ただし，RAM 領域については，アドレス範囲が 0xF0000 ～ 0xFFFFF である内部 RAM のみサポートします。

注 3. 配置の境界制限について，デフォルトの設定を 64KB - 1 境界とします。

注 4. 16 ビット・データへのアクセスを保証するため，整列条件を「2」とします。

注 5. 絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。

注 6. 特別なセクションのため，セクション名の変更を禁止とし，セクション名固定とします。

注 7. オプション・バイト，およびオンチップ・デバッグ，セキュリティ ID の配置先アドレス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。

- セクション名の指定がないセクション定義には，アセンブラが再配置属性ごとにデフォルトのセクション名を与えます。
  次に，付与されるセクション名を示します。

再配置属性	セクション名
CALLT0	.callt
TEXT	.text
TEXTF	.textf
TEXTF_UNIT64KP	.textf_unit64kp
CONST	.const
CONSTF	.constf
AT △アドレス	.text_AT 開始アドレス
OPT_BYTE	.option_byte[注]
SECUR_ID	.security_id[注]
FLASH_SECUR_ID	.flash_security_id[注]

注 特別なセクションのため，セクション名の変更を禁止とし，セクション名固定とします。

上記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。

- 再配置属性の指定がないセクション定義は，再配置属性 “TEXT” となります。
- 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（@，_，.）

［使用例］

TEXT 属性のセクション ".text" を定義

```
.text   .CSEG   TEXT
        NOP
```

TEXTF_UNIT64KP 属性のセクション "unit" を定義

```
unit    .CSEG   TEXTF_UNIT64KP
        MOV     A, !LABEL
```

AT 属性のセクション "EX" を 0x00200 番地指定で定義
セクション名は "EX_AT200" となります。

```
EX      .CSEG   AT 0x00200
        .DS     2
```

オプションバイト用のセクションを定義

```
.CSEG   OPT_BYTE
.DB     0xFF
```

.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。

[詳細説明]

- プログラム中でデータ定義を行う部分を .DSEG 疑似命令により定義します。
  本疑似命令は，次にセクション定義疑似命令が記述されるまで有効です。
- .DSEG 疑似命令は，オペランド欄に DATA_AT，BSS_AT を指定することにより開始アドレスを指定することができます。
  また，.ORG 疑似命令により開始アドレスを指定することもできます。
  このときのセクション名は，指定されたセクション名＋ “_AT” ＋指定アドレス（ただし，prefix（0x，0X）および suffix（h，H）が付かない大文字の 16 進表記）となります。
- 次に，.DSEG の指定可能な再配置属性を示します。

表 5.17 .DSEG の再配置属性

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注1]
SDATA	SDATA	.sdata	初期値を持つデータ用のセクションを saddr 領域[注2] 内で先頭が偶数番地になるように配置します。	2
SBSS	SBSS	.sbss	初期値を持たないデータ用のセクションを saddr 領域[注2] 内で先頭が偶数番地になるように配置します。	2
DATA	DATA	.data	初期値を持つデータ用のセクションを RAM 領域[注2] 内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
BSS	BSS	.bss	初期値を持たないデータ用のセクションを RAM 領域[注2] 内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
DATAF	DATAF	.dataf	初期値を持つデータ用のセクションを先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
BSSF	BSSF	.bssf	初期値を持たないデータ用のセクションを先頭が偶数番地になるように，64KB - 1 境界[注3] にまたがらないように配置します。	2
DATA_AT △アドレス	DATA_AT 絶対式[注4]	なし	初期値を持つデータ用のセクションを指定番地に配置します。	1（固定）

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値[注 1]
BSS_AT △アドレス	BSS_AT 絶対式[注 4]	なし	初期値を持たないデータ用のセクションを指定番地に配置します。	1（固定）

注 1. 整列条件は .ALIGN 疑似命令にて変更することができます。

注 2. コード・フラッシュ領域，ミラー領域，RAM 領域，saddr 領域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。ただし，RAM 領域については，アドレス範囲が 0xF0000 ～ 0xFFFFF である内部 RAM のみサポートします。

注 3. 配置の境界制限について，デフォルトの設定を 64KB - 1 境界とします。

注 4. 絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。

- 初期値を持たないデータ用のセクション定義中に，初期値を指定する疑似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。
- セクション名の指定がないセクション定義には，アセンブラが再配置属性ごとにデフォルトのセクション名を与えます。
  次に，付与されるセクション名を示します。

再配置属性	セクション名
SDATA	.sdata
SBSS	.sbss
DATA	.data
BSS	.bss
DATAF	.dataf
BSSF	.bssf
DATA_AT △アドレス	.data_AT 開始アドレス
BSS_AT △アドレス	.bss_AT 開始アドレス

上記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。

- 再配置属性の指定がないセクション定義は，再配置属性 “DATA” となります。
- 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（@，_，.）

［使用例］

DATA 属性のセクション “.data” を定義

```
.data   .DSEG   DATA
        .DS     4
```

SDATA 属性のセクション “_S” を定義

```
_S      .DSEG   SDATA
        .DS     4
```

DATA_AT 属性のセクション “EX” を 0xff000 番地指定で定義
セクション名は “EX_ATFF000” となります。

```
EX      .DSEG   DATA_AT 0xff000
        .DS     2
```

.BSEG

アセンブラにビット・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*]	.BSEG	[*再配置属性*]	[; *コメント*]

[機能]

- .BSEG 疑似命令は，アセンブラにビット・セクションの開始を指示します。
- .BSEG 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，ビット・セクションに属します。

[詳細説明]

- プログラム中でビットのデータ定義を行う部分を本疑似命令により定義します。
  本疑似命令は，次にセクション定義疑似命令が記述されるまで有効です。
- 本疑似命令は，オペランド欄に BIT_AT を指定することにより開始アドレスを指定することができます。
  このときのセクション名は，指定されたセクション名＋ “_AT” ＋指定アドレス（ただし，prefix（0x，0X）および suffix（h，H）が付かない大文字の 16 進表記）となります。
- 次に，.BSEG の指定可能な再配置属性を示します。

表 5.18 .BSEG の再配置属性

再配置属性	記述形式	デフォルトのセクション名	説明	整列条件のデフォルト値
SBSS_BIT	SBSS_BIT	.sbss_bit	初期値を持たないビット用のセクションを saddr 領域[注 1] 内で先頭が偶数番地になるように配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を SBSS として扱います。[注 4] [注 5]	2
BSS_BIT	BSS_BIT	.bss_bit	初期値を持たないビット用のセクションを RAM 領域[注 1] 内の 0xF0000 ～ 0xFFFFF 番地内に，先頭が偶数番地になるように，64KB - 1 境界[注 2] にまたがらないように配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を BSS として扱います。[注 4] [注 6]	2
BIT_AT △アドレス	BIT_AT 絶対式[注 3]	なし	初期値を持たないビット用のセクションを指定番地に配置します。最適化リンカは本セクションをバイト単位で結合し，再配置属性を BSS_AT として扱います。[注 4] [注 7]	1（固定）

注 1. コード・フラッシュ領域，ミラー領域，RAM 領域，saddr 領域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。ただし，RAM 領域については，アドレス範囲が 0xF0000 ～ 0xFFFFF である内部 RAM のみサポートします。

注 2. 配置の境界制限について，デフォルトの設定を 64KB - 1 境界とします。

注 3. 絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。

注 4. 再配置属性 SBSS と SBSS_BIT，再配置属性 BSS と BSS_BIT，再配置属性 BSS_AT と BIT_AT の同名セクションは，アセンブラ内部で連続した 1 つのセクションとして処理します。

注 5. セクションを再配置属性 SBSS としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 SBSS として配置します。

注 6. セクションを再配置属性 BSS としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 BSS として配置します。

注 7. セクションを再配置属性 BSS_AT としてオブジェクト・ファイルに出力し，最適化リンカは再配置属性 BSS_AT として配置します。

- 本疑似命令で定義したセクション中に記述できる命令は，.DBIT，.EQU，.SET，.PUBLIC，.EXTBIT，.EXTERN 疑似命令およびマクロ呼び出しです。
  他の機械語命令や疑似命令を記述した場合はエラーとなります。
- セクション名の指定がないセクション定義には，アセンブラが再配置属性ごとにデフォルトのセクション名を与えます。
  次に，付与されるセクション名を示します。

再配置属性	セクション名
SBSS_BIT	.sbss_bit
BSS_BIT	.bss_bit
BIT_AT △アドレス	.bit_AT 開始アドレス

上記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。

- 再配置属性の指定がないセクション定義は，再配置属性 “SBSS_BIT” となります。
- 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（@，_，.）

［使用例］

SBSS_BIT 属性のセクション “.sbss_bit” を定義

```
.sbss_bit   .BSEG   SBSS_BIT
sym01       .DBIT
```

BSS_BIT 属性のセクション “_B” を定義

```
_B      .BSEG   BSS_BIT
        .DBIT
sym02   .DBIT
```

BIT_AT 属性のセクション “EX” を 0xffe20 番地指定で定義
セクション名は “EX_ATFFE20” となります。

```
EX      .BSEG   BIT_AT 0xffe20
sym03   .DBIT
        .DBIT
```

.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。

[詳細説明]

- .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名（ただし，絶対アドレス形式セクションであれば “_AT” 以降は含まない）” ＋ “_AT” ＋ “ 指定アドレス（ただし，prefix（0x，0X）および suffix（h，H）が付かない大文字の 16 進表記）” 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。
- オペランドの値は「(a) 絶対式」に従います。絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。
- 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名のセクション定義がすでに存在する場合，および本疑似命令で指定したセクションのアドレスが同一モジュール内のほかの絶対アドレス形式セクションの配置範囲内である場合はエラーとなります。

[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
LAB1:   MOV         A, !LABEL
        .ORG        0x30                ;My_text.AT30 を 0x30 に配置する
        MOV         A,!LABEL
```

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         A,!LABEL                    ;My_text_AT50 に配置する
```

絶対アドレス形式セクションに .ORG を記述した場合はそのセクション名の “_AT” より前の部分に新たに “_AT”+ 指定アドレスを付加したセクション名となります。

```
.SECTION        My_text, AT     0x20
NOP                                     ;My_text_AT20 に配置する
.ORG            0x50
MOV             A,!LABEL                ;My_text_AT50 に配置する
```

.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
[*ラベル* :]	.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，またはデータ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。

[詳細説明]

- 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。
- オペランドの絶対式は「(a) 絶対式」の定義に従います。絶対式として不正な記述をした場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合はエラーとなります。
- 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 疑似命令の行のオフセットから，本疑似命令によって指定されたオフセットまでの間の領域の初期値は “0x0” です。
- .OFFSET 疑似命令はセクションの再配置属性に BSS が付く場合は記述できません。エラーとなります。

[使用例]

```
.SECTION    My_data, text
.OFFSET     0x12
MOV         A, B            ;オフセットは 0x12 になる
```

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 5.19 シンボル定義疑似命令

疑似命令	概要
.EQU	絶対式の値を持つネームの定義
.SET	絶対式の値を持つネームの定義

.EQU

絶対式の値を持つ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	.EQU	*絶対式* [. *ビット位置*]	[; *コメント*]
ネーム	.EQU	PSW[. *ビット位置*]	[; *コメント*]

[機能]

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]

- .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ネームは .PUBLIC 疑似命令で外部定義することができます。
- オペランドには以下を記述できます。
  - 絶対式
    絶対式の定義は「(a) 絶対式」に従います。
  - PSW
    PSW が指定された場合，値は 0xFFFFA となります。

  不正な記述をした場合はエラーとなります。
- オペランドに指定できる値の範囲は 0x00000000 ～ 0xFFFFFFFF，ビット位置に指定できる値の範囲は 0 ～ 7 です。
- オペランドにリロケータブル項は記述できません。また，オペランドに演算結果がリロケータブル項となる演算子は記述できません。

[使用例]

定数式の指定

```
SYM1    .EQU    10
        MOV     A, #SYM1
```

シンボルの指定

```
SYM2    .EQU    0xFFE20
BSYM1   .EQU    SYM2.1
        SET1    BSYM1
```

PSW の指定

```
SYM3    .EQU    PSW
        MOV     SYM3, #10
```

SFR 略号の指定

```
SYM4    .EQU    P0
        MOV     SYM4, #2
```

.SET

絶対式の値を持つ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]

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]

- .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ネームです。
- .SET 疑似命令で生成したネームは .PUBLIC 疑似命令で外部定義することができません。
- オペランドの記述は「(a) 絶対式」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ます。
- オペランドの絶対式に指定できる値の範囲は 0x00000000 ～ 0xFFFFFFFF です。
- .EQU 疑似命令とは異なり，ビット位置指定を持つシンボルは定義できません。
- オペランドに演算結果がリロケータブル項となる演算子は記述できません。

[使用例]

定数式の指定

```
SYM1    .SET    10
        MOV     A, #SYM1
```

SFR 略号の指定

```
SYM2    .SET    P0
        MOV     SYM2, #2
```

5.2.4 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 5.20 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	2 バイト領域を初期化
.DB4	4 バイト領域を初期化
.DB8	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
.DBIT	1 ビットのビット領域を確保
.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	{ *式* |" *文字列定数* "}[, ...]	[; *コメント*]

[機能]

- 1 バイト領域を初期化します。

[詳細説明]

- バイト領域を初期化します。

  (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。式の値の下位 24 ビットが 0x0 ～ 0xFF の間でない場合は，エラーを出力します。
  リロケータブルなシンボルや外部参照名を含んだ式が記述することができます。

  (b) 文字列定数
  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，必要なバイト数分が確保されます。

- .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]

```
        .DSEG   DATA
LABEL:  .DB     10      ; 1 バイトの領域を 10 で初期化する
        .DB     "ABC"   ; 3 バイトの領域を文字列 "ABC"(0x41，0x42，0x43) で初期化する
```

.DB2

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	*式* [, ...]	[; *コメント*]

[機能]

- 2 バイト領域を初期化します。

[詳細説明]

- 2 バイト領域を初期化します。
  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 0x0000 ～ 0xFFFF の間でなければなりません。式の値の下位 24 ビットが 0x0000 ～ 0xFFFF の間でない場合は，エラーを出力します。
  リロケータブルなシンボルや外部参照名を含んだ式が記述することができます。
- オペランドに文字列定数は，記述できません。
- .DB2 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]

```
        .DSEG   DATA
LABEL:  .DB2    0x1234  ; 2 バイトの領域を 0x34，0x12 で初期化する
```

.DB4

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	*式* [, ...]	[; *コメント*]

[機能]

- 4 バイト領域を初期化します。

[詳細説明]

- 4 バイト領域を初期化します。
  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 0x00000000 ～ 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 4 バイト・データとして確保されます。
  リロケータブルなシンボルや外部参照名を含んだ式が記述することができます。
- オペランドに文字列定数は，記述できません。
- .DB4 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]

```
        .DSEG   DATA
LABEL:  .DB4    0x12345678  ; 4 バイトの領域を 0x78，0x56，0x34，0x12 で初期化する
```

.DB8

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*]

[機能]

- 8 バイト領域を初期化します。

[詳細説明]

- 8 バイト領域を初期化します。
  絶対式の値は，8 バイト・データとして確保されます。したがって，絶対式の値は 0x00000000 00000000 ～ 0xFFFFFFFF FFFFFFFF 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 8 バイト・データとして確保されます。
- オペランドには「(a) 絶対式」が記述可能です。
  ただし，.DB8 疑似命令では式の各項を 64 ビットとして扱います。このため，.DB8 疑似命令では .EQU 疑似命令で定義されたネーム（32 ビット）を負の数として使用することはできません。
- オペランドに文字列定数は，記述できません。
- .DB8 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。

[使用例]

```
        .DSEG   DATA
LABEL:  .DB8    0x1234567890ABCDEF  ; 8 バイトの領域を
                                    ; 0xEF，0xCD，0xAB，0x90，0x78，0x56，0x34，0x12
                                    ; で初期化する
```

.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*]

[機能]

- オペランドで指定されたバイト数分の領域を確保します。

[詳細説明]

- 初期値を持たないデータ用のセクションでは，オペランドで指定されたバイト数分の領域を確保します。
  その他のセクションでは，オペランドで指定されたバイト数分の領域を 0 で初期化して確保します。
  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。
- サイズには「(a) 絶対式」が記述可能です。
- サイズの記述が不正な場合，または 0x00000 ～ 0xFFFFF の範囲を越える場合は，エラーが出力されます。
- ラベルを記述した場合は，確保した領域の先頭アドレスを値に持つシンボルとして定義されます。

[使用例]

```
        .DSEG   DATA
AREA1:  .DS     4       ; 4 バイトの領域を 0 で初期化して確保する
        .DSEG   BSS
AREA2:  .DS     8       ; 8 バイトの領域を確保する
```

.DBIT

1 ビットのビット領域を確保します。

[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
[*ネーム*]	.DBIT		[; *コメント*]

[機能]

- 1 ビットのビット領域を確保します。

[詳細説明]

- 本疑似命令はビット・セクション内でのみ記述できます。
- ビット・セクション内で 1 ビットのビット領域を確保する際に使用します。
- 確保したビット領域の内容は不定です。
- シンボル欄にネームを記述した場合，アドレスとビット位置を値として持つビット・シンボルとして定義します。

[使用例]

```
        .BSEG
BSYM1   .DBIT   ; 1 ビットのビット領域を確保する
        .DBIT   ; 1 ビットのビット領域を確保する
BSYM2   .DBIT   ; 1 ビットのビット領域を確保する
        .CSEG
        SET1    BSYM1
        CLR1    BSYM2
```

.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*	[; *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，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発生した領域は 0 で埋めます（ただし，BSS が付く再配置属性のセクションは除きます）。
- 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，エラーを出力します。
- .ALIGN 疑似命令は，現在のセクションに対する本疑似命令を指定したモジュー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，最適化リンカで配置した後のアドレスを整列するものではありません。

[使用例]

```
        .CSEG   TEXT
        .DS     1       ;OFFSET 0x0
        .ALIGN  2       ;OFFSET 0x2 ;1byte パディング
LABEL:                  ;OFFSET 0x2 ; 最適化リンカでの配置時にセクション先頭を奇数番地に
                                    ; 配置すると整列条件は 2 にならない
                                    ;（先頭アドレス 0x1 なら 0x3 に配置される）
```

```
        .SECTION  D1, DATA
        .DB       1
        .ALIGN    4
        .DB       2
        .ALIGN    6
        .DB       3
        ; セクションの整列条件はセクション内に含まれる整列条件の最小公倍数となるため，
        ; D1 セクションの整列条件は 12 となる。
```

5.2.5 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 5.21 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シンボルを宣言
.EXTBIT	本モジュールで参照するほかのモジュールのビット・シンボルを宣言
.WEAK 【V1.11 以降】	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

.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	*シンボル名*	[; *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。

[詳細説明]

- 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。その際，ワーニングになったシンボル名をメッセージ中に出力します。
  どのモジュールにも定義がない場合，リンク時にエラーとなります。
- .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。
- 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) セクション名
  (c) マクロ名

[使用例]

```
          .PUBLIC  PSYM01
          .PUBLIC  PSYM02
PSYM01:   .DB      0x10
PSYM02:
          MOV      A, #0x4D
```

.EXTERN

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。

[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	*シンボル名*	[; *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。

[詳細説明]

- オペランドに指定したシンボルを本モジュールで参照できるようになります。
- オペランドには，次のものを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) セクション名
  (c) マクロ名
- .EXTERN 疑似命令で宣言されたシンボルを本モジュール中で参照しなくても，エラーにはなりません。
- .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。

[使用例]

参照側プログラム

```
        .EXTERN  PSYM01
        .EXTERN  PSYM02
        MOV      A, ES:!PSYM01
        BR       !PSYM02
```

定義側プログラム

```
        .PUBLIC  PSYM01
        .PUBLIC  PSYM02
PSYM01: .DB      0x10
PSYM02: MOV      A, #0x4D
```

.EXTBIT

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* :]	.EXTBIT	*シンボル名*	[; *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ビット・シンボルを宣言します。

[詳細説明]

- オペランドに指定したビット・シンボルを本モジュールで参照できるようになります。
- オペランドには，次のものを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) セクション名
  (c) マクロ名
- .EXTBIT 疑似命令で宣言されたビット・シンボルを本モジュール中で参照しなくても，エラーにはなりません。
- .EXTBIT 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。

[使用例]

```
        .EXTBIT  EBIT01
        .EXTBIT  EBIT02
        MOV1     EBIT01, CY
        AND1     CY, EBIT02
```

.WEAK 【V1.11 以降】

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* :]	.WEAK	*シンボル名*	[; *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。

[詳細説明]

- 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。
- .PUBLIC 疑似命令と次の点で異なります。
  - 異なるモジュールに存在する同名シンボルに対して，それぞれに .PUBLIC 疑似命令を指定した場合，リンク時にエラーになります。
  - 異なるモジュールに存在する同名シンボルに対して，一つに .PUBLIC 疑似命令を，他に .WEAK 疑似命令を指定した場合，エラーにならず，.PUBLIC 疑似命令を指定したモジュールがリンクされます。
- .WEAK 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。
- 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) セクション名

(c) マクロ名

5.2.6 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 5.22 コンパイラ出力疑似命令

疑似命令	概要
.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
.VECTOR	割り込みベクタテーブルを生成
.ALIAS 【V1.11.00 以降】	シンボルの情報を設定
.TYPE 【V1.11.00 以降】	シンボルの情報を設定

.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。

[注意事項]

- .LINE 疑似命令が扱う情報は，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。ユーザは使用しないでください。

.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 = 絶対式*	[; *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，0x0000 ～ 0xFFFE の範囲の 2 の倍数のみとし，それ以外を指定した場合はその定義は無効となります。
- .STACK 疑似命令が扱う情報は，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，領域確保疑似命令（.DB/.DB2/.DB4/.DB8/.DS）
  - マクロ疑似命令（.MACRO/.IRP/.REPT/.LOCAL/.ENDM）
  - 外部定義疑似命令（.PUBLIC）【V1.04 以降】
- inline_asm 指定関数内の .PUBLIC 疑似命令には，inline_asm 指定関数内で定義したラベル以外を記述できません。記述した場合はエラーとなります。
- ._LINE_TOP 疑似命令が扱う情報は，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。ユーザは使用しないでください。

._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，領域確保疑似命令（.DB/.DB2/.DB4/.DB8/.DS）
  - マクロ疑似命令（.MACRO/.IRP/.REPT/.LOCAL/.ENDM）
  - 外部定義疑似命令（.PUBLIC）【V1.04 以降】
- inline_asm 指定関数内の .PUBLIC 疑似命令には，inline_asm 指定関数内で定義したラベル以外を記述できません。記述した場合はエラーとなります。
- ._LINE_END 疑似命令が扱う情報は，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。ユーザは使用しないでください。

.VECTOR

割り込みベクタテーブルを生成します。

[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
シンボル名	.VECTOR	*ベクタテーブル配置アドレス*	[; *コメント*]

[機能]

- .VECTOR 疑似命令は割り込みベクタテーブルを生成します。

[詳細説明]

- 割り込みハンドラにベクタテーブル配置アドレスを設定します。
- シンボル名には割り込み発生時の分岐先ラベルを指定します。
- ベクタテーブル配置アドレスとして指定可能なのは，0x00000 ～ 0x0007E の範囲にある偶数アドレスで，それ以外の値を指定した場合はエラーとなります。
- １つの関数に対し複数のアドレスの設定が可能です。
- .VECTOR 疑似命令が扱う情報は，C 言語ソース・プログラムの #pragma interrupt 指定に対してコンパイラが出力するベクタテーブル情報です。本疑似命令が含まれる場合（C 言語ソース・プログラムで #pragma interrupt 指定を行った場合）は，スタートアップ・ルーチン等のアセンブリ・ソース・プログラム中でベクタテーブルを定義する際に，セクション定義疑似命令ではなく本疑似命令を使用する必要があります。
- .VECTOR 疑似命令で記述されたベクタテーブルとセクション定義疑似命令で記述されたベクタテーブルが混在する場合，最適化リンカでエラーとなります。

[使用例]

割り込みハンドラ intfunc に対して複数のアドレス（0x00008，0x0000a）を設定

```
_intfunc        .VECTOR 0x00008
_intfunc        .VECTOR 0x0000a
.SECTION        .text, TEXT
_intfunc:
```

割り込みハンドラ intfunc2 に対して複数のアドレス（0x00020，0x00022，0x00024，0x00026，0x00028）を設定

```
ADDR            .SET    0x00020
                .REPT   5
_intfunc2       .VECTOR ADDR
ADDR            .SET    ADDR + 2
                .ENDM
.SECTION        .text, TEXT
_intfunc2:
```

.ALIAS 【V1.11.00 以降】

シンボルの情報を設定します。

[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
	.ALIAS	*別名 , シンボル名*	[; *コメント*]

[詳細説明]

- シンボルの別名を生成します。
- 生成した別名を .PUBLIC 疑似命令に指定することが可能です。
- シンボル名に，モジュール内に存在しないラベルを指定した場合はエラーになります。
- . 本疑似命令は，オペランドのシンボル名に指定したシンボルの定義より後に記述する必要があります。

[使用例]

次のように記述すると，別名の symA のみがモジュール外から参照できるようになります。

```
symB:
.PUBLIC symA
.ALIAS symA, symB
```

```
SECTION=
SYMBOL ADDR SIZE INFO COUNTS OPT

 symA
   00000120 0 none ,g 0
 symB
   00000120 0 none ,l 0
```

.TYPE 【V1.11.00 以降】

シンボルの情報を設定します。

[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
	.TYPE	*シンボル名，シンボル種別，サイズ指定*	[; *コメント*]

[詳細説明]

- シンボルの種別と，サイズ情報を指定します。
- シンボル種別には，"FUNCTION" または "OBJECT" を指定可能です。
- シンボル名に，ファイル内に存在しないラベルを指定した場合はエラーになります。

[使用例]

```
.TYPE symA, FUNCTION, 8
symA:
```

設定した情報はリンク・マップで確認できます。

```
SYMBOL ADDR SIZE INFO COUNTS OPT
  symA
   00000000 8 func ,l 0
```

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にもなります。

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 5.23 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ネームをマクロ名とした，マクロを定義
.LOCAL	指定したシンボル名を所属するマクロ・ボディ内でのみ有効なローカル・シンボルとして定義
.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絶対式の値分だけ，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ネームをマクロ名とした，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
ネーム	.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ネームをマクロ名とした，マクロの定義を行います。
  このマクロ名を参照した場合（以下，「マクロ呼び出し」と呼ぶ），その位置にこのマクロ名に対応するマクロ本体が記述されたものとして扱います（以下，「マクロ展開」と呼ぶ）。

[詳細説明]

- .MACRO 疑似命令に対応する .ENDM 疑似命令が，同一ファイル内に存在しない場合はエラーが出力されます。
  例えば，.MACRO，または .ENDM のどちらか一方がインクルード・ファイル中に存在する場合はエラーとなります。
- 仮パラメータの最大数は利用可能なメモリ量に依存します。
- マクロ呼び出しにおいて，余分な実パラメータがある場合は警告が出力されます。
- マクロ定義より先にマクロ呼び出しを行った場合，エラーが出力されます。
- 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，エラーが出力されます。
- 同名の仮パラメータが複数ある場合はエラーが出力されます。
- マクロ呼び出しにおいて，実パラメータとしては数値定数，またはシンボルのいずれかが記述可能であり，そうでない場合はエラーが出力されます。
- 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。

[使用例]

```
ADMAC   .MACRO  PARA1, PARA2   ; マクロ定義
        MOV     A, #PARA1
        ADD     A, #PARA2
        .ENDM

        ADMAC   0x10, 0x20     ; マクロ呼び出し
```

.LOCAL

指定したシンボル名を所属するマクロ・ボディ内でのみ有効なローカル・シンボルとして定義します。

[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
[*ラベル* :]	.LOCAL	*シンボル名* [, ...]	[; *コメント*]

[機能]

- 指定したシンボル名を所属するマクロ・ボディ内でのみ有効なローカル・シンボルとして定義します。

[詳細説明]

- 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。
- シンボル名の最大数は利用可能なメモリ量に依存します。
- .LOCAL 疑似命令はマクロ定義中，および C ソースの “#pragma inline_asm” 中のみ記述できます。
- .LOCAL 疑似命令は，オペランド欄で指定したシンボルを参照する前に記述しなければなりません。マクロ・ボディ先頭での記述を推奨します。
- 1 つのモジュール中で .LOCAL 疑似命令により定義するシンボル名は，すべて別名でなければなりません。各マクロ・ボディ中で使用するローカル・シンボル名に同一名は使用できません。
- .LOCAL 疑似命令で定義したシンボルをマクロ外から参照することはできません。
- シンボルとして予約語を記述することはできません。ただし，ユーザ定義シンボルと同じシンボルを記述した場合は，.LOCAL 疑似命令の定義が優先されます。
- マクロ定義の仮引数と同名のシンボルを，そのマクロ定義中に .LOCAL 疑似命令でローカル・シンボル定義した場合はエラーとなります。

[使用例]

```
m1      .MACRO  par
        .LOCAL  AA, BB
AA:     .DB4    AA
BB:     .DB4    par
        .ENDM
m1      10
m1      20
```

展開すると次のようになります。

```
.LL00000000:    .DB4    .LL00000000
.LL00000001:    .DB4    10
.LL00000002:    .DB4    .LL00000002
.LL00000003:    .DB4    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絶対式の値分だけ，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絶対式の値分だけ，アセンブラが繰り返し展開します。

[詳細説明]

- .REPT 疑似命令に対応する .ENDM 疑似命令が，同一ファイル内に存在しない場合は，エラーとなります。
- 絶対式の評価結果が負になった場合は，エラーとなります。

[使用例]

```
    .REPT    3
    INC      B
    DEC      C
    .ENDM
```

アセンブル後は以下のように展開します。

```
    INC      B
    DEC      C
    INC      B
    DEC      C
    INC      B
    DEC      C
```

.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	*仮パラメータ* [*実パラメータ* [, ...]]	[; *コメント*]
	:		
	.ENDM		[; *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。

[詳細説明]

- .IRP 疑似命令に対応する .ENDM 疑似命令が，同一ファイル内に存在しない場合は，エラーとなります。
- 実パラメータの最大数は利用可能なメモリ量に依存します。

[使用例]

```
    .IRP    PAR 0x10, 0x20, 0x30
    ADD     A, #PAR
    MOV     [DE], A
    .ENDM
```

アセンブル後は以下のように展開します。

```
    ADD     A, #0x10
    MOV     [DE], A
    ADD     A, #0x20
    MOV     [DE], A
    ADD     A, #0x30
    MOV     [DE], A
```

.EXITM

.EXITM 疑似命令を囲んでいる最も内側の .REPT，.IRP 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
	.EXITM		[; *コメント*]

[機能]

- .EXITM 疑似命令は，本疑似命令を囲んでいる最も内側の .REPT，.IRP 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]

- .EXITM 疑似命令が .REPT，.IRP 疑似命令に囲まれていない場合，エラーとなります。
- .EXITM 疑似命令を囲む条件アセンブル制御命令は，.ENDM 疑似命令までの間に記述できません。（【V1.01 のみ】）

[使用例]

```
    .REPT   3
    .REPT   2
    INC     B
    .EXITM
    .ENDM
    DEC     C
    .ENDM
```

アセンブル後は以下のように展開します。

```
    INC     B
    DEC     C
    INC     B
    DEC     C
    INC     B
    DEC     C
```

.EXITMA

.EXITMA 疑似命令を囲んでいる最も外側の .REPT，.IRP 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
	.EXITMA		[; コメント]

[機能]

- .EXITMA 疑似命令を囲んでいる最も外側の .REPT，.IRP 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]

- .EXITMA 疑似命令が .REPT，.IRP 疑似命令に囲まれていない場合，エラーとなります。
- .EXITMA 疑似命令を囲む条件アセンブル制御命令は，.ENDM 疑似命令までの間に記述できません。(【V1.01 のみ】)

[使用例]

```
    .REPT   3
    .REPT   2
    INC     B
    .EXITMA
    .ENDM
    DEC     C
    .ENDM
```

アセンブル後は以下のように展開します。

```
    INC     B
```

.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。

[詳細説明]

- .ENDM 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，エラーとなります。

[使用例]

.MACRO - .ENDM

```
ADMAC   .MACRO   PARA1, PARA2
        MOV      A, #PARA1
        ADD      A, #PARA2
        .ENDM
```

.REPT - .ENDM

```
       .REPT     3
       INC       B
       DEC       C
       .ENDM
```

.IRP - .ENDM

```
       .IRP      PAR 0x10, 0x20, 0x30
       ADD       A, #PAR
       MOV       [DE], A
       .ENDM
```

5.2.8 分岐疑似命令

分岐疑似命令は，コンパイラが生成しアセンブラが命令展開を行うものです。
分岐疑似命令には，次のものがあります。

表 5.24 分岐疑似命令

疑似命令	概要
.Bcond	本疑似命令に対して，アセンブラは命令展開を行う

.B*cond*

.B*cond* 疑似命令に対して，アセンブラは命令展開を行います。

備考 .BT，.BF，.BC，.BNC，.BZ，.BNZ，.BH，.BNH をまとめて，.B*cond* としています。

［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
	.B*cond*	*ラベル*	[；*コメント*]

- 上記形式の場合，.B*cond* として指定可能なのは .BC，.BNC，.BZ，.BNZ，.BH，.BNH です。
  また，ラベルとして指定可能なのは !LABEL，$!LABEL，!!LABEL の形式です。

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.B*cond*	*ビット項，ラベル*	[；*コメント*]

- 上記形式の場合，.B*cond* として指定可能なのは .BT，.BF です。
  また，ラベルとして指定可能なのは !LABEL，$!LABEL，!!LABEL の形式です。

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.B*cond*	*ラベル 1，ラベル 2*	[；*コメント*]

- 上記形式の場合，.B*cond* として指定可能なのは .BC，.BNC，.BZ，.BNZ，.BH，.BNH です。
  また，ラベル 1，ラベル 2 としてとして指定可能なのは以下の組み合わせです。

ラベル 1	ラベル 2
!LABEL1	$LABEL2
!LABEL1	!LABEL2
$!LABEL1	$LABEL2
$!LABEL1	$!LABEL2
$!LABEL1	!!LABEL2
!!LABEL1	$LABEL2
!!LABEL1	$!LABEL2
!!LABEL1	!!LABEL2

シンボル欄	ニモニック欄	オペランド欄	コメント欄
	.B*cond*	*ビット項，ラベル 1，ラベル 2*	[；*コメント*]

- 上記形式の場合，.B*cond* として指定可能なのは .BT，.BF です。
  また，ラベル 1，ラベル 2 としてとして指定可能なのは以下の組み合わせです。

ラベル 1	ラベル 2
!LABEL1	!LABEL2
$!LABEL1	$!LABEL2
$!LABEL1	!!LABEL2

ラベル 1	ラベル 2
!!LABEL1	$!LABEL2
!!LABEL1	!!LABEL2

［機能］

- .B*cond* 疑似命令に対して，アセンブラは命令展開を行います。
- .B*cond* 疑似命令はコンパイラが生成することを前提としており，ユーザは使用しないでください。

［詳細説明］

- .B*cond* 疑似命令はコンパイラが生成し，アセンブラは以下のように展開します。
  N*cond* は *cond* と逆の条件を表します。

展開前	展開後
.B*cond* *ラベル*	SKN*cond* BR *ラベル*
.B*cond* *ビット項*	BN*cond* *ビット項*, $temp BR *ラベル* temp:
.B*cond* *ラベル 1*, *ラベル 2*	SKN*cond* BR *ラベル 1* BR *ラベル 2*
.B*cond* *ビット項*, *ラベル 1*, *ラベル 2*	B*cond* *ビット項*, $temp BR *ラベル 2* temp: BR *ラベル 1*

5.2.9 機械語命令セット

下記の命令は RL78 の命令セットにありませんが，実在する命令に置き換えて解釈します。それ以外の RL78 の命令セットにない命令を記述した場合は，エラーとします。

命令オペランド置換前	命令オペランド置換後
MOV [DE],#byte	MOV [DE+0],#byte
MOV [HL],#byte	MOV [HL+0],#byte
MOV ES:[DE],#byte	MOV ES:[DE+0],#byte
MOV ES:[HL],#byte	MOV ES:[HL+0],#byte
MOVS [HL],X	MOVS [HL+0],X
MOVS ES:[HL],X	MOVS ES:[HL+0],X
CMPS X,[HL]	CMPS X,[HL+0]
CMPS X,ES:[HL]	CMPS X,ES:[HL+0]
ADDW AX,[HL]	ADDW AX,[HL+0]
ADDW AX,ES:[HL]	ADDW AX,ES:[HL+0]
SUBW AX,[HL]	SUBW AX,[HL+0]
SUBW AX,ES:[HL]	SUBW AX,ES:[HL+0]
CMPW AX,[HL]	CMPW AX,[HL+0]
CMPW AX,ES:[HL]	CMPW AX,ES:[HL+0]
INC [HL]	INC [HL+0]
INC ES:[HL]	INC ES:[HL+0]
DEC [HL]	DEC [HL+0]
DEC ES:[HL]	DEC ES:[HL+0]
INCW [HL]	INCW [HL+0]
INCW ES:[HL]	INCW ES:[HL+0]
DECW [HL]	DECW [HL+0]
DECW ES:[HL]	DECW ES:[HL+0]

5.3 制御命令

制御命令とは，アセンブラの動作に対し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。
次に，制御命令の種類を示します。

表 5.25　　制御命令一覧

制御命令の種類	制御命令
ファイル入力制御命令	INCLUDE，BINCLUDE
ミラー元領域参照制御命令	MIRROR
アセンブラ制御命令	NOWARNING，WARNING
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 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 5.26 ファイル入力制御命令

制御命令	概要
INCLUDE	アセンブリ言語ソース・ファイルの入力
BINCLUDE	バイナリ・ファイルの入力

INCLUDE

アセンブリ言語ソース・ファイルを入力します。

[指定形式]

```
[ △ ]$[ △ ]INCLUDE[ △ ]( ファイル名 )[ △ ][; コメント]
[ △ ]$[ △ ]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。

[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) ソース・ファイルのあるフォルダ
  (c) 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とします。

BINCLUDE

バイナリ・ファイルを入力します。

[指定形式]

```
[ △ ]$[ △ ]BINCLUDE[ △ ]( ファイル名 )[ △ ][; コメント ]
[ △ ]$[ △ ]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]

- 本制御命令は，バイナリ・ファイルの内容全体を扱います。
- 指定したバイナリ・ファイルは，$INCLUDE 制御命令と同じ優先順位で検索されます。
- 指定したバイナリ・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。

5.3.3 ミラー元領域参照制御命令

ミラー元領域参照制御命令を用いることにより，ミラー元領域アドレスに配置されるセクションに属する外部参照名を参照できるようになります。
ミラー元領域参照制御命令には，次のものがあります。

表 5.27 ミラー元領域参照制御命令

制御命令	概要
MIRROR	外部参照名がミラー元領域に配置されていると宣言

MIRROR

外部参照名がミラー元領域に配置されていると宣言します。

[指定形式]

```
[ △ ]$[ △ ]MIRROR[ △ ] 外部参照名 [ △ ][; コメント ]
```

[機能]

- 外部参照名がミラー元領域に配置されるものとみなし，そのデータを参照する命令を生成します。

[詳細説明]

- ミラー元領域に配置される外部参照名を参照する場合に利用します。

5.3.4 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 5.28 アセンブラ制御命令

制御命令	概要
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

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]

- 本制御命令以降の命令に対し，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］

- 本制御命令以降の命令に対し，警告メッセージを出力します（NOWARNING 指定を解除）。

5.3.5 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 5.29 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	絶対式による制御（真のときアセンブル）
IFN	絶対式による制御（偽のときアセンブル）
ELSEIF	絶対式による制御（真のときアセンブル）
ELSEIFN	絶対式による制御（偽のときアセンブル）
ELSE	絶対式／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］

- スイッチ名で指定したシンボル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。

- スイッチ名で指定したシンボル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。

［詳細説明］

- スイッチ名記述上の規則は，シンボル記述上の規則「(3) シンボル」と同じです。
- 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。

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]

- スイッチ名で指定したシンボル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。
- スイッチ名で指定したシンボル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。

[詳細説明]

- スイッチ名記述上の規則は，シンボル記述上の規則「(3) シンボル」と同じです。
- 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。

IF

絶対式による制御（真のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IF △絶対式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対式が真（≠ 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。

[詳細説明]

- 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。

IFN

絶対式による制御（偽のときアセンブル）をします。

[指定形式]

```
[ △ ]$[ △ ]IFN △絶対式 [ △ ][; コメント ]
```

[機能]

- オペランドで指定した絶対式が真（≠ 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。

[詳細説明]

- 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。

ELSEIF

絶対式による制御（真のときアセンブル）をします。

［指定形式］

```
[ △ ]$[ △ ]ELSEIF △絶対式 [ △ ][; コメント]
```

［機能］

- オペランドで指定した絶対式が真（≠ 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。

ELSEIFN

絶対式による制御（偽のときアセンブル）をします。

［指定形式］

```
[ △ ]$[ △ ]ELSEIFN △絶対式 [ △ ][; コメント]
```

［機能］

- オペランドで指定した絶対式が真（≠ 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。

ELSE

絶対式／シンボルによる制御をします。

［指定形式］

[△]$[△]ELSE[△][; 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。

ENDIF

制御範囲の終わりを示します。

［指定形式］

```
[ △ ]$[ △ ]ENDIF[ △ ][; 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。

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。

```
ADDINT8 .MACRO  PARA1, PARA2    ; 次の 2 つの文がマクロ本体
        MOV     A, #PARA1
        ADD     A, #PARA2
        .ENDM
```

上記を定義したあと，次のように記述した場合，「0x10 と 0x20 を加算する」というコードに置き換えられます。

```
ADDINT8 0x10, 0x20
```

したがって，次のようなコードに展開されます。

```
MOV     A, #0x10
ADD     A, #0x20
```

5.4.3 マクロ定義のネスティング

.MACRO，.REPT，.IRP 疑似命令と対応する .ENDM 疑似命令までの間に，.MACRO 疑似命令を記述することはできません。

5.4.4 マクロ参照のネスティング

マクロ参照のネスティング・レベルの最大数は 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。

5.4.5 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 “?” について説明します。

- 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，コンカティネート記号自身は消滅します。
- 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・シンボルをシンボル中に連結する前に評価します。
- 注意事項
  - 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。

5.4.6 エラー処理

.MACRO，.REPT，.IRP 疑似命令と .ENDM 疑似命令の対応については，以下のように扱います。

- .MACRO，.REPT，.IRP 疑似命令に対して，ソース・プログラムの最後まで対応する .ENDM 疑似命令が見つからない場合，ソース・プログラムの最後にエラーが出力されます。
- .ENDM 疑似命令に対し，対応する .MACRO，.REPT，.IRP 疑似命令が見つからない場合，エラーが出力されます。
- .MACRO 疑似命令の定義行にエラーがあった場合，その行を無視して処理を続けます。したがって，マクロ名は定義されず，参照時にエラーが出力されます。

5.5 SFR 略号，および拡張 SFR 略号の利用

アセンブラではデバイス・ファイルを読み込んだ場合，SFR 略号，および拡張 SFR 略号を記述することができます。
アセンブラで用いる SFR 略号，および拡張 SFR 略号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。
SFR 略号，および拡張 SFR 略号は定数（その SFR のアドレス）のように扱われますが，下記の点に注意が必要です。

- SFR 略号，および拡張 SFR 略号は大文字／小文字を区別しません。
- SFR 略号，および拡張 SFR 略号をシンボルとして定義することはできません。
- 命令オペランド中の拡張 SFR 略号の先頭には「!」が必要です。
  アセンブラ移行支援機能オプション（-convert_asm）指定時は，「!」を省略できます。

また，「5.1.16 オペランドの特性」の sfr, sfrp, addr16 の説明も参照してください。

5.6 予 約 語

アセンブラには予約語が存在します。予約語をネーム，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。予約語を定義した場合は，メッセージが出力されます。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

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 5.30 アセンブラ生成シンボル

シンボル名	説明
.LL00000000 ～ .LLFFFFFFFF	.LOCAL 疑似命令生成ローカル・シンボル
.text_AT *開始アドレス*	.CSEG 疑似命令生成セクション名
.data_AT *開始アドレス*	.DSEG 疑似命令生成セクション名
.bss_AT *開始アドレス*	
.bit_AT *開始アドレス*	.BSEG 疑似命令生成セクション名
セクション名 _AT *開始アドレス*	.ORG 疑似命令生成セクション名
.BR_TEMP@*n*	分岐命令展開用ラベル
.LM*n*_*n* （*n*：0 ～ 4294967294 ）	アセンブラ・デバッグ情報用シンボル （セクション，関数の先頭／末尾） - 例：.LM0_1
.G*n* （*n*：0 ～ 4294967294 ）	アセンブラ・デバッグ情報用シンボル （.debug_info，line，frame,loc の先頭／末尾）
.Gar*n* （*n*：0 ～ 4294967294 ）	アセンブラ・デバッグ情報用シンボル （.debug_pubnames のセクション先頭／末尾）
.Gpu*n* （*n*：0 ～ 4294967294 ）	アセンブラ・デバッグ情報用シンボル （.debug_aranges のセクション先頭／末尾）
@$IMM_ *定数値*	定数値を示すローカル・シンボル【V1.02 以降】

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。
この章ではメモリ配置の単位であるセクションについて説明します。

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。
セクション名はソース・ファイル内で指定できます。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 セクション名

以下に，予約されているセクションの名前とそれらの再配置属性を示します。

表 6.1 予約セクション

デフォルト・セクション名	再配置属性	内容
.callt0	CALLT0	callt 関数呼び出しのテーブル用セクション
.text	TEXT	コード部用セクション（near 領域配置）
.textf	TEXTF	コード部用セクション（far 領域配置）
.textf_unit64kp	TEXTF_UNIT64KP	コード部用セクション（セクションを先頭が偶数番地になるように，64KB-1 境界にまたがらないように配置）
.const	CONST	ROM データ（near 領域配置）（ミラー領域内）
.constf	CONSTF	ROM データ（far 領域配置）
.data	DATA	near 初期化データ用セクション（初期値あり）
.dataf	DATAF	far 初期化データ用セクション（初期値あり）
.sdata	SDATA	初期化データ用セクション（初期値あり，saddr 配置変数）
.bss	BSS	データ領域用セクション（初期値なし，near 領域配置）
.bssf	BSSF	データ領域用セクション（初期値なし，far 領域配置）
.sbss	SBSS	データ領域用セクション（初期値なし，saddr 配置変数）
.option_byte	OPT_BYTE	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用セクション
.security_id	SECUR_ID	セキュリティ ID 指定専用セクション
.flash_security_id	FLASH_SECUR_ID	フラッシュ・プログラマ・セキュリティ ID 指定専用セクション
.vect< ベクタテーブル・アドレス >	AT	割り込みベクタテーブル -split_vect オプションを指定したとき，.vect< ベクタテーブル・アドレス > でセクションを生成します。ベクタテーブル・アドレスは 16 進数表記になります。

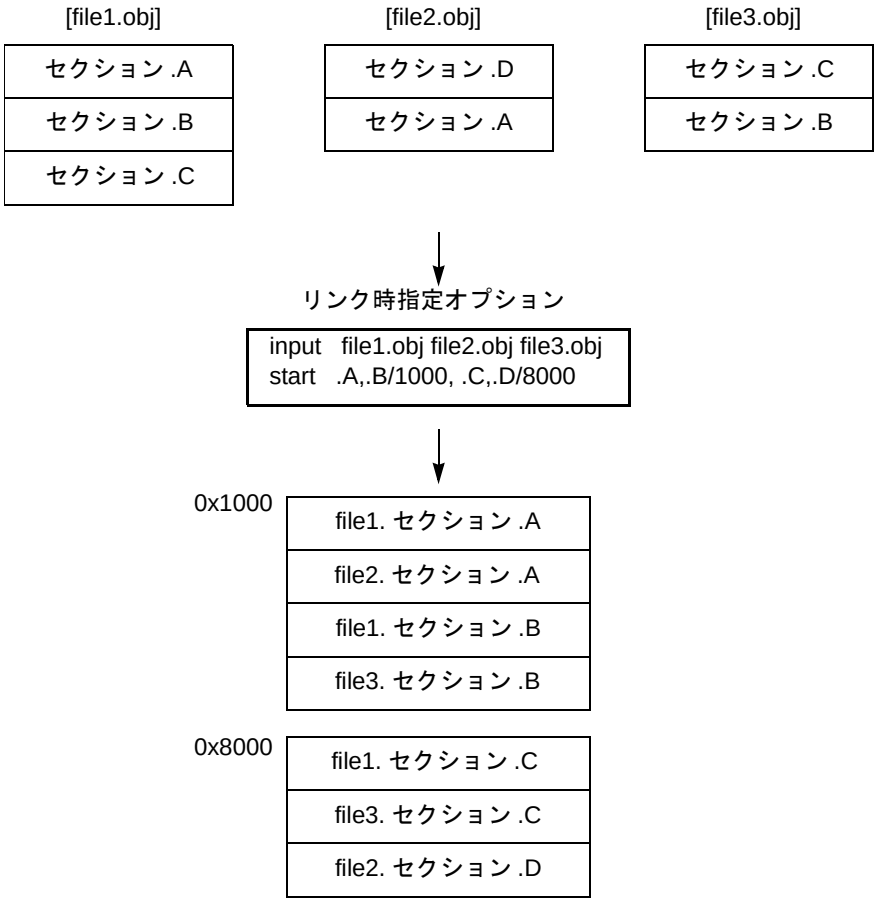
6.1.2 # セクションの結合

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。

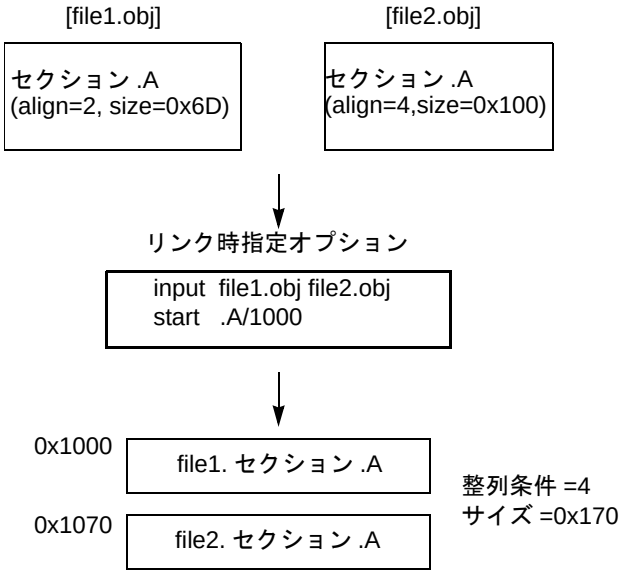
備考 -start オプションの詳細は「-STARt」を参照してください。

(1) -start オプションによるセクション配置

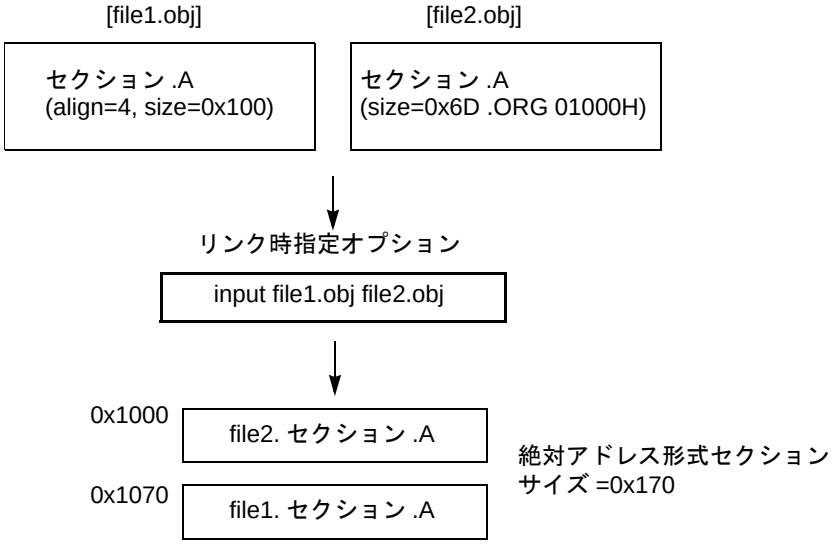
(a) 異なるファイルの同名セクションは，ファイルの入力順に連続して割り付けます。



(b) 整列条件数の異なる同名セクションは，整列条件調整後に結合します。セクションの整列条件数は大きい方に合わせます。

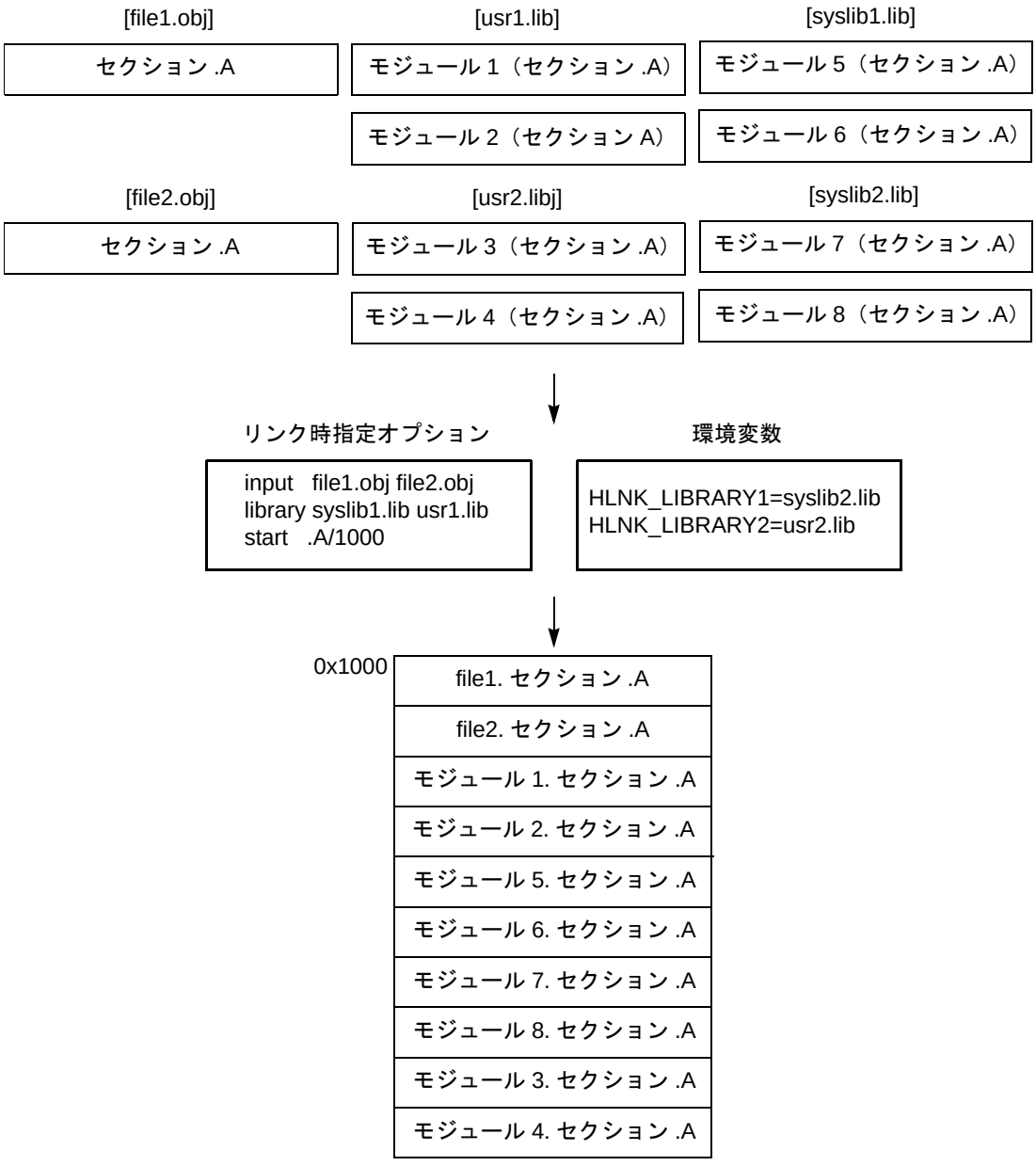


(c) 同名セクションに絶対アドレス形式と相対アドレス形式が含まれている場合，絶対アドレス形式セクションの後に相対アドレス形式セクションを結合します。



(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

最適化リンカは，リンクの処理において，各出力セクションの先頭と末尾を指すシンボル，またはオプション指定によりシンボルを生成します。

6.2.1 オプション指定に関係なく生成するシンボル

最適化リンカは，.data や .sdata などのセクションの先頭と末尾を指すシンボルを生成します。
先頭シンボル：__s セクション名
末尾シンボル：__e セクション名
例えば，.data セクションの先頭シンボルは __s.data，末尾シンボルは __e.data というシンボル名になります。なお，最適化リンカが生成するシンボルと同名のシンボルが定義されている場合は，シンボルを生成しません。

6.2.2 オプション指定により生成するシンボル

最適化リンカは，オプション指定により以下に示すシンボルを生成します。

表 6.2 オプション指定により生成するシンボル

オプション	シンボル名	説明
-hide	$CNCL_*n* n：1 ～ 4294967295 例）$CNCL_1	出力ファイル内のローカルシンボル名を $CNCL_*n* に変換します。
-user_opt_byte -ocdbg	.option_byte	デバイス・ファイルが指定され，かつ，ソース・ファイルにセクション名 “.option_byte” が存在しない場合，セクション名 “.option_byte” を生成します。なお，上記セクションには，ユーザ・オプション・バイト値，およびオンチップ・デバッグ動作の制御値が設定されます。
-security_id	.security_id	デバイス・ファイルが指定され，かつ，ソース・ファイルにセクション名 “. security_id” が存在しない場合，セクション名 “.security_id” を生成します。なお，上記セクションには，セキュリティ ID 値が設定されます。
-device	__STACK_ADDR_START __STACK_ADDR_END	デバイス・ファイルから取得した RAM 領域内で利用していない連続領域を探し，__STACK_ADDR_START，および __STACK_ADDR_END に以下の値を設定します。 __STACK_ADDR_START：領域の最大アドレス + 1 __STACK_ADDR_END： 領域の最小アドレス ただし，-SELF オプション，-SELFW オプション，-OCDTR オプション，-OCDTRW オプション，-OCDHPI オプション，または -OCDHPIW オプションが指定されている場合，saddr 領域を除いた RAM 領域から設定します。
	__RAM_ADDR_START __RAM_ADDR_END	デバイス・ファイルから RAM 領域を取得し，__RAM_ADDR_START，および __RAM_ADDR_END に以下の値を設定します。 __RAM_ADDR_START：RAM 領域の先頭アドレス __RAM_ADDR_END： RAM 領域の終了アドレス + 1
-debug_monitor	.monitor1 .monitor2	デバイス・ファイルが指定され，かつソース・ファイルにセクション名 ”.monitor1”，および ”.monitor2” が存在しない場合，セクション名 ”.monitor1”，および ”.monitor2” を生成します。
-rrm	.rrm	【V1.13 以降】ソース・ファイルにセクション名 ”.rrm” が存在しない場合，セクション名 ”.rrm” を生成します。

7. ライブラリ関数仕様

この章では，CC-RL 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-RL 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 【V1.03 以降】 （calloc，free，malloc，realloc を除く）	rl78nm4s.lib rl78nm4s99.lib 【V1.07 以降】	拡張命令なし，演算器なし 8 ビット CPU/16 ビット CPU 用
	rl78cm4s.lib rl78cm4s99.lib 【V1.07 以降】	乗除・積和演算器使用 16 ビット CPU 用
	rl78em4s.lib rl78em8s.lib rl78em4s99.lib 【V1.07 以降】 rl78em8s99.lib 【V1.07 以降】	乗除算拡張命令使用 16 ビット CPU 用
【V1.03 以降】 標準ライブラリ （calloc，free，malloc，realloc）	malloc_n.lib	通常用 malloc 系ライブラリ
	malloc_s.lib	セキュリティ機能用 malloc 系ライブラリ 【Professional 版のみ】
ランタイム・ライブラリ	rl78nm4r.lib	拡張命令なし，演算器なし 8 ビット CPU/16 ビット CPU 用
	rl78cm4r.lib	乗除・積和演算器使用 16 ビット CPU 用
	rl78em4r.lib rl78em8r.lib	乗除算拡張命令使用 16 ビット CPU 用
スタートアップ・ルーチン	cstart.asm	スタートアップ・ルーチン

変数ポインタを引数に持つ標準ライブラリ関数に対して far ポインタを渡したい場合は，関数名が “_COM_” から始まる far 変数ポインタ用の関数をユーザが呼ぶ必要があります。なお，-far_rom オプション指定時は，ヘッダ・ファイルの関数マクロが有効になり，自動的に far 変数ポインタ用の関数が呼ばれます。

7.2 ライブラリ・ファイル命名規則

標準ライブラリを，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。

ランタイム・ライブラリ関数は，浮動小数点演算や整数演算を行うときに，CC-RL が自動的に呼び出すルーチンです。

また，リンク・オプション（-library）で，これらのライブラリを参照するようにします。同一のプロジェクト内で使用するライブラリ・ファイルの種類は統一しなければなりません。

ライブラリ・ファイルの命名規則は，次のようになっています。

【V1.03 以降】

malloc 系のライブラリ関数は通常用とセキュリティ機能用とで機能が異なるため，他の標準ライブラリとは別のライブラリとなります。malloc 系ライブラリは RL78-S1，RL78-S2，RL78-S3 共通です。

rl78<*muldiv*><*memory_model*><*float*><*standard/runtime*><*lang*>.lib malloc_<*secure*>.lib （malloc 系ライブラリ）【V1.03 以降】

<*muldiv*>

n ：拡張命令なし，演算器なし（RL78-S1 コア／ RL78-S2 コア用）

c ：乗除・積和演算器使用（RL78-S2 コア用）

e ：乗除算拡張命令使用（RL78-S3 コア用）
<*memory_model*>
m ：スモール／ミディアム・モデル用
<*float*>
4 ：単精度浮動小数点数
8 ：倍精度浮動小数点数（乗除算拡張命令ありデバイスのみ対応）
<*standard/runtime*>
s ：標準ライブラリ
r ：ランタイム・ライブラリ
<*secure*>
n ：通常用
s ：セキュリティ機能用 【Professional 版のみ】
<*lang*>
なし: C90 用
99 ：C99 用 【V1.07 以降】

7.3 ライブラリおよびスタートアップ・ルーチンの配置先

ライブラリ関数のコード用セクションは，全領域に配置可能です。スタートアップ・ルーチンは 0x00000 ～ 0x0FFFF 番地にのみ配置可能です。

セクション名	再配置属性	内容
.RLIB	TEXTF	ランタイム・ライブラリ　コード用セクション
.SLIB	TEXTF	標準ライブラリ　コード用セクション
.data .bss .constf	DATA BSS CONSTF	標準ライブラリ　データ用セクション
.text	TEXT	スタートアップ・ルーチン

7.4 ヘッダ・ファイル

CC-RL 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・変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
inttypes.h（C99 のみ）【V1.07 以降】	最大幅整数型のためのヘッダ・ファイル
iso646.h（C99 のみ）【V1.06 以降】	代替つづりマクロのためのヘッダ・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
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
stdbool.h（C99 のみ）【V1.06 以降】	論理型と論理値のためのヘッダ・ファイル
stddef.h	共通の定義のためのヘッダ・ファイル
stdint.h	指定した幅の整数型のためのヘッダ・ファイル
stdio.h	標準入出力のためのヘッダ・ファイル
stdlib.h	一般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	初期化処理のためのヘッダ・ファイル

7.5 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。
C90 と C99 とで仕様が異なる場合は，C90 用ライブラリ関数は（C90），C99 用ライブラリ関数は（C99）と記載しています。【V1.07 以降】

標準・数学ライブラリ使用時は，必ずヘッダ・ファイルをインクルードしてください。
-far_rom オプション指定時は，ポインタの near/far 属性のインタフェースが合うように，ヘッダ・ファイル内で関数マクロ化して far 変数ポインタ用の関数を呼びます。
ポインタを引数に持つ場合，標準入出力関数のフォーマット指定，文字列操作関数のトークン分割については far ポインタ固定とします。それ以外はメモリ・モデル，および -far_rom オプションに依存します。
スモール／ミディアム・モデル指定時に far ポインタを引数に渡す場合は，far 変数ポインタ用の関数を呼びます。
倍精度指定時に float 型で数学関数を使用したい場合は，関数名の末尾に “f” が付いた float 型用の数学関数を呼びます。

7.5.1 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>
void    assert(int expression);
```

[詳細説明]

プログラム中に診断機能を付け加えます。
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。
-lang=c99 指定時には，関数名も診断情報に含まれます。【V1.06 以降】

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。

7.5.2 文字操作関数

文字操作関数として，以下のものがあります。

表 7.4 文字操作関数

関数名	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
isblank 【V1.07 以降】	スペースまたはタブであるかを判定（C99）
iscntrl	制御文字であるかを判定
isdigit	10 進数であるかを判定
isgraph	スペース以外の表示文字であるかを判定
islower	英小文字であるかを判定
isprint	表示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
isupper	英大文字であるかを判定
isxdigit	16 進数であるかを判定
toascii	ASCII コードへ変換
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（引数が英大文字以外のときそのまま）

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 __far       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。

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 __far       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。

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 __far       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。

isblank 【V1.07 以降】

スペースまたはタブ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 __far isblank(int c);（C99）
```

［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。

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 __far       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。

isdigit

10 進数の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 __far       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。

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 __far       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。

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 __far       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。

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 __far       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。

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 __far       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。

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 __far       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。

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 __far       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。

isxdigit

16 進数の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 __far       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。

toascii

ASCII コードへ変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int __far       toascii(int c);
```

[戻り値]

c に対して下位 7 ビットでマスクした値を返します。

[詳細説明]

c の値を ASCII コードに変換します。ASCII コードの範囲（ビット 0 ～ 6）以外のビット（ビット 7 ～ 15）は 0 にします。

tolower

英大文字から英小文字に変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int __far       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，ほかの文字はすべてそのままにします。

toupper

英小文字から英大文字に変換します。

[所属]

標準ライブラリ

[指定形式]

```
#include <ctype.h>
int __far       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，ほかの文字はすべてそのままにします。

7.5.3 最大幅整数型のための関数

最大幅整数型のための関数として，以下のものがあります。

表 7.5 最大幅整数型のための関数

関数名	概要
imaxabs 【V1.07 以降】	絶対値（intmax_t 型）を出力（C99）
imaxdiv 【V1.07 以降】	除算（intmax_t 型）（C99）
strtoimax 【V1.07 以降】	文字列を整数（intmax_t 型）へ変換し，最終文字列へのポインタを格納（C99）
strtoumax 【V1.07 以降】	文字列を整数（uintmax_t 型）へ変換し，最終文字列へのポインタを格納（C99）

imaxabs 【V1.07 以降】

絶対値（intmax_t 型）を求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <inttypes.h>
intmax_t __far imaxabs(intmax_t j);（C99）
```

［戻り値］

j の絶対値（*j* の大きさ），|*j*| を返します。imaxabs の入力値が負の最小値である場合は，同値を返します。

［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。

imaxdiv 【V1.07 以降】

intmax_t 型の除算を行い，商と余りを求めます。

［所属］

標準ライブラリ

［指定形式］

#include <inttypes.h>
imaxdiv_t __far imaxdiv(intmax_t *numer*, intmax_t *denom*);（C99）

［戻り値］

除算の結果を格納した構造体を返します。0 で割った場合，商 quot は -1，剰余 rem は *numer* が設定されます。

［詳細説明］

intmax_t 型の値を除算する場合に使用します。
分子 *numer* を分母 *denom* で割ったその商 quot と剰余 rem を算出し，その 2 つの整数を次に示す構造体 imaxdiv_t のメンバとして格納します。

```
typedef struct {
    intmax_t quot;
    intmax_t rem;
} imaxdiv_t;
```

割り切れない場合，結果の商は，代数的な商に最も近くそれより絶対値が小さい整数となります。

strtoimax 【V1.07 以降】

文字列を整数（intmax_t 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 <inttypes.h>
intmax_t __far strtoimax(const char __near * restrict nptr, char __near * __near * restrict endptr, int base);（C99）
intmax_t __far _COM_strtoimax_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base);（C99）
```

［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。
オーバフローが生じる場合，INTMAX_MAX，または INTMAX_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を intmax_t 型の表現に変換します。*base* が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

strtoumax 【V1.07 以降】

文字列を整数（uintmax_t 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 <inttypes.h>
uintmax_t __far strtoumax(const char __near * restrict nptr, char __near * __near * restrict endptr, int base);（C99）
uintmax_t __far _COM_strtoumax_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base);（C99）
```

［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。
オーバフローが生じる場合，UINTMAX_MAX を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を uintmax_t 型の表現に変換します。*base* が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

7.5.4 数学関数

数学関数として，以下のものがあります。

表 7.6 数学関数

関数／マクロ名	概要
fpclassify 【V1.08 以降】	NaN，無限大，正規化数，非正規化数，0 に分類（C99）
isfinite 【V1.08 以降】	有限（0，非正規化数，正規化数）かどうかを判定（C99）
isinf 【V1.08 以降】	無限大であるかどうかを判定（C99）
isnan 【V1.08 以降】	NaN かどうかを判定（C99）
isnormal 【V1.08 以降】	正規化数（0，非正規化数，無限大，NaN のいずれでもない）かどうかを判定（C99）
signbit 【V1.08 以降】	符号が負かどうかを判定（C99）
acos	逆余弦
acosf	逆余弦
acosl 【V1.08 以降】	逆余弦（C99）
asin	逆正弦
asinf	逆正弦
asinl 【V1.08 以降】	逆正弦（C99）
atan	逆正接
atanf	逆正接
atanl 【V1.08 以降】	逆正接（C99）
atan2	逆正接（y / x）
atan2f	逆正接（y / x）
atan2l 【V1.08 以降】	逆正接（y / x）（C99）
cos	余弦
cosf	余弦
cosl 【V1.08 以降】	余弦（C99）
sin	正弦
sinf	正弦
sinl 【V1.08 以降】	正弦（C99）
tan	正接
tanf	正接
tanl 【V1.08 以降】	正接（C99）
acosh 【V1.08 以降】	双曲線逆余弦（C99）
acoshf 【V1.08 以降】	双曲線逆余弦（C99）
acoshl 【V1.08 以降】	双曲線逆余弦（C99）
asinh 【V1.08 以降】	双曲線逆正弦（C99）
asinhf 【V1.08 以降】	双曲線逆正弦（C99）

関数／マクロ名	概要
asinhl 【V1.08 以降】	双曲線逆正弦（C99）
atanh 【V1.08 以降】	双曲線逆正接（C99）
atanhf 【V1.08 以降】	双曲線逆正接（C99）
atanhl 【V1.08 以降】	双曲線逆正接（C99）
cosh	双曲線余弦
coshf	双曲線余弦
coshl 【V1.08 以降】	双曲線余弦（C99）
sinh	双曲線正弦
sinhf	双曲線正弦
sinhl 【V1.08 以降】	双曲線正弦（C99）
tanh	双曲線正接
tanhf	双曲線正接
tanhl 【V1.08 以降】	双曲線正接（C99）
exp	指数関数（自然対数）
expf	指数関数（自然対数）
expl 【V1.08 以降】	指数関数（自然対数）（C99）
frexp	浮動小数点数を正規化した数と 2 のべき乗に分割
frexpf	浮動小数点数を正規化した数と 2 のべき乗に分割
frexpl 【V1.08 以降】	浮動小数点数を正規化した数と 2 のべき乗に分割（C99）
ldexp	浮動小数点数と 2 の整数べき乗を乗算
ldexpf	浮動小数点数と 2 の整数べき乗を乗算
ldexpl 【V1.08 以降】	浮動小数点数と 2 の整数べき乗を乗算（C99）
log	対数関数（自然対数）
logf	対数関数（自然対数）
logl 【V1.08 以降】	対数関数（自然対数）（C99）
log10	対数関数（10 を底）
log10f	対数関数（10 を底）
log10l 【V1.08 以降】	対数関数（10 を底）（C99）
log1p 【V1.08 以降】	実引数に 1 を加えた値の対数関数（自然対数）（C99）
log1pf 【V1.08 以降】	実引数に 1 を加えた値の対数関数（自然対数）（C99）
log1pl 【V1.08 以降】	実引数に 1 を加えた値の対数関数（自然対数）（C99）
modf	浮動小数点数を整数部と小数部に分割
modff	浮動小数点数を整数部と小数部に分割
modfl 【V1.08 以降】	浮動小数点数を整数部と小数部に分割（C99）
scalbn 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）

関数／マクロ名	概要
scalbnf 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）
scalbnl 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）
scalbln 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）
scalblnf 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）
scalblnl 【V1.09 以降】	浮動小数点数と FLT_RADIX のべき乗を乗算（C99）
fabs	絶対値関数
fabsf	絶対値関数
fabsl 【V1.08 以降】	絶対値関数（C99）
pow	べき乗関数
powf	べき乗関数
powl 【V1.08 以降】	べき乗関数（C99）
sqrt	平方根関数
sqrtf	平方根関数
sqrtl 【V1.08 以降】	平方根関数（C99）
ceil	浮動小数点数以上の最小の整数値
ceilf	浮動小数点数以上の最小の整数値
ceill 【V1.08 以降】	浮動小数点数以上の最小の整数値（C99）
floor	浮動小数点数以下の最大の整数値
floorf	浮動小数点数以下の最大の整数値
floorl 【V1.08 以降】	浮動小数点数以下の最大の整数値（C99）
nearbyint 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99)
nearbyintf 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99)
nearbyintl 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99）
rint 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99）
rintf 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99）
rintl 【V1.09 以降】	現在の丸め方向に従った，浮動小数点形式の整数値への丸め（C99）
lrint 【V1.09 以降】	現在の丸め方向に従った，long 型整数値への丸め（C99）
lrintf 【V1.09 以降】	現在の丸め方向に従った，long 型整数値への丸め（C99）
lrintl 【V1.09 以降】	現在の丸め方向に従った，long 型整数値への丸め（C99）
llrint 【V1.09 以降】	現在の丸め方向に従った，long long 型整数値への丸め（C99）
llrintf 【V1.09 以降】	現在の丸め方向に従った，long long 型整数値への丸め（C99）
llrintl 【V1.09 以降】	現在の丸め方向に従った，long long 型整数値への丸め（C99）
round 【V1.09 以降】	浮動小数点形式の整数値への丸め（C99）
roundf 【V1.09 以降】	浮動小数点形式の整数値への丸め（C99）
roundl 【V1.09 以降】	浮動小数点形式の整数値への丸め（C99）

関数／マクロ名	概要
lround 【V1.09 以降】	long 型整数値への丸め（C99）
lroundf 【V1.09 以降】	long 型整数値への丸め（C99）
lroundl 【V1.09 以降】	long 型整数値への丸め（C99）
llround 【V1.09 以降】	long long 型整数値への丸め（C99）
llroundf 【V1.09 以降】	long long 型整数値への丸め（C99）
llroundl 【V1.09 以降】	long long 型整数値への丸め（C99）
trunc 【V1.09 以降】	切り捨てた整数値への丸め（C99）
truncf 【V1.09 以降】	切り捨てた整数値への丸め（C99）
truncl 【V1.09 以降】	切り捨てた整数値への丸め（C99）
fmod	剰余関数
fmodf	剰余関数
fmodl 【V1.08 以降】	剰余関数（C99）
copysign 【V1.09 以降】	与えられた絶対値と符号からなる値を生成（C99）
copysignf 【V1.09 以降】	与えられた絶対値と符号からなる値を生成（C99）
copysignl 【V1.09 以降】	与えられた絶対値と符号からなる値を生成（C99）
nan 【V1.09 以降】	文字列を NaN に変換（C99）
nanf 【V1.09 以降】	文字列を NaN に変換（C99）
nanl 【V1.09 以降】	文字列を NaN に変換（C99）
fdim 【V1.09 以降】	正の差の計算（C99）
fdimf 【V1.09 以降】	正の差の計算（C99）
fdiml 【V1.09 以降】	正の差の計算（C99）
fmax 【V1.09 以降】	大きい方の値を取得（C99）
fmaxf 【V1.09 以降】	大きい方の値を取得（C99）
fmaxl 【V1.09 以降】	大きい方の値を取得（C99）
fmin 【V1.09 以降】	小さい方の値を取得（C99）
fminf 【V1.09 以降】	小さい方の値を取得（C99）
fminl 【V1.09 以降】	小さい方の値を取得（C99）
isgreater 【V1.09 以降】	最初の引数が 2 番目の引数より大きいか判定（C99）
isgreaterequal 【V1.09 以降】	最初の引数が 2 番目の引数以上か判定（C99）
isless 【V1.09 以降】	最初の引数が 2 番目の引数より小さいか判定（C99）
islessequal 【V1.09 以降】	最初の引数が 2 番目の引数以下か判定（C99）
islessgreater 【V1.09 以降】	最初の引数が 2 番目の引数より小さい，または大きいか判定（C99）
isunordered 【V1.09 以降】	順序付けられていないかどうか判定（C99）

fpclassify 【V1.08 以降】

実引数の値を NaN，無限大，正規化数，非正規化数，0 に分類します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int fpclassify( 実浮動小数点型 x); (C99)
```

［戻り値］

実引数の値が NaN である場合，FP_NAN を返します。
実引数の値が無限大である場合，FP_INFINITE を返します。
実引数の値が正規化数である場合，FP_NORMAL を返します。
実引数の値が非正規化数である場合，FP_SUBNORMAL を返します。
実引数の値が 0 である場合，FP_ZERO を返します。
返却する各マクロの値は math.h を参照してください。

［詳細説明］

実引数の値を NaN，無限大，正規化数，非正規化数，0 に分類します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

isfinite 【V1.08 以降】

有限（0，非正規化数，正規化数）かどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int isfinite( 実浮動小数点型 x); (C99)
```

［戻り値］

実引数が有限（0，非正規化数，正規化数）である場合，0 以外の値を返します。
実引数が無限大または NaN である場合，0 を返します。

［詳細説明］

実引数が有限（0，非正規化数，正規化数）かどうかを判定します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

isinf 【V1.08 以降】

無限大であるかどうかを判定します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
int isinf( 実浮動小数点型 x); (C99)
```

[戻り値]

実引数が正または負の無限大である場合，0 以外の値を返します。
それ以外の場合は 0 を返します。

[詳細説明]

実引数が正または負の無限大であるかどうかを判定します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

isnan 【V1.08 以降】

NaN かどうかを判定します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
int isnan( 実浮動小数点型 x); (C99)
```

[戻り値]

実引数が NaN である場合，0 以外の値を返します。
それ以外の場合は 0 を返します。

[詳細説明]

実引数が NaN かどうかを判定します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

isnormal 【V1.08 以降】

正規化数（0，非正規化数，無限大，NaN のいずれでもない）かどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int isnormal( 実浮動小数点型 x); (C99)
```

［戻り値］

実引数が正規化数である場合，0 以外の値を返します。
実引数が 0，非正規化数，無限大，NaN の場合は 0 を返します。

［詳細説明］

実引数が正規化数かどうかを判定します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

signbit 【V1.08 以降】

符号が負かどうかを判定します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
int signbit( 実浮動小数点型 x); (C99)
```

[戻り値]

実引数の符号が負である場合，0 以外の値を返します。
実引数の符号が正である場合，0 を返します。

[詳細説明]

実引数の符号が負かどうかを判定します。
実引数に浮動小数点型以外の型を渡した場合，動作を保証しません。

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 __far 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, π] の範囲です。
x が [-1, 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 を設定します。

[詳細説明]

x の逆余弦（アークコサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ 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 <math.h>
float   __far   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, π] の範囲です。
x が [-1, 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 を設定します。

[詳細説明]

x の逆余弦（アークコサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ で指定します。

acosl 【V1.08 以降】

逆余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far acosl(long double x); (C99)
```

[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, π] の範囲です。
x が [-1, 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 を設定します。

[詳細説明]

x の逆余弦（アークコサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ 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 __far 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, π / 2] の範囲です。
x が [-1, 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 を設定します。

[詳細説明]

x の逆正弦（アークサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ 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 <math.h>
float   __far   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, π / 2] までの範囲です。
x が [-1, 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 を設定します。

[詳細説明]

x の逆正弦（アークサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ で指定します。

asinl 【V1.08 以降】

逆正弦を求めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far asinl(long double x); (C99)
```

［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, π / 2] までの範囲です。
x が [-1, 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 を設定します。

［詳細説明］

x の逆正弦（アークサイン）を求めます。*x* は，$-1 \leqq x \leqq 1$ で指定します。

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 __far 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, π / 2] の範囲です。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
x が -0 である場合，-0 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 の逆正接（アークタンジェント）を求めます。

atanf

逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   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, π / 2] の範囲です。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
x が -0 である場合，-0 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 の逆正接（アークタンジェント）を求めます。

atanl 【V1.08 以降】

逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far atanl(long double x); (C99)
```

[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, π / 2] の範囲です。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
x が -0 である場合，-0 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 の逆正接（アークタンジェント）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 __far 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, π] までの範囲です。
x と *y* のどちらかが非数，または *x* と *y* がともに 0，または *x* と *y* がともに ± ∞の場合，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，または消滅して 0 になった場合，グローバル変数 errno にマクロ ERANGE を設定します。

［詳細説明］

y / *x* の逆正接（アークタンジェント）を求めます。その際，両方の実引数の符号を用いて返却値の象限を決定します。

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 <math.h>
float   __far   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, π] までの範囲です。
x と *y* のどちらかが非数，または *x* と *y* がともに 0，または *x* と *y* がともに ± ∞の場合，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，または消滅して 0 になった場合，グローバル変数 errno にマクロ ERANGE を設定します。

［詳細説明］

y / *x* の逆正接（アークタンジェント）を求めます。その際，両方の実引数の符号を用いて返却値の象限を決定します。

atan2l 【V1.08 以降】

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>
long double __far atan2l(long double *y*, long double *x*); (C99)

［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, π] までの範囲です。
x と *y* のどちらかが非数，または *x* と *y* がともに 0，または *x* と *y* がともに ± ∞の場合，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，または消滅して 0 になった場合，グローバル変数 errno にマクロ ERANGE を設定します。

［詳細説明］

y / *x* の逆正接（アークタンジェント）を求めます。その際，両方の実引数の符号を用いて返却値の象限を決定します。

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  __far   cos(double x);
```

[戻り値]

x の余弦を返します。
x が非数，または ± ∞である場合，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 の余弦を求めます。角度はラジアン単位で指定します。

cosf

余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   cosf(float x);
```

[戻り値]

x の余弦を返します。
x が非数，または ± ∞である場合，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 の余弦を求めます。角度はラジアン単位で指定します。

cosl 【V1.08 以降】

余弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far cosl(long double x); (C99)
```

[戻り値]

x の余弦を返します。
x が非数，または ± ∞である場合，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 の余弦を求めます。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  __far   sin(double x);
```

[戻り値]

x の正弦を返します。
x が非数，または ± ∞である場合，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。

sinf

正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   sinf(float x);
```

[戻り値]

x の正弦を返します。
x が非数，または ± ∞である場合，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。

sinl 【V1.08 以降】

正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far sinl(long double x); (C99)
```

[戻り値]

x の正弦を返します。
x が非数，または ± ∞である場合，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。

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  __far   tan(double x);
```

[戻り値]

x の正接を返します。
x が非数，または ± ∞である場合，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。

tanf

正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   tanf(float x);
```

[戻り値]

x の正接を返します。
x が非数，または ± ∞である場合，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。

tanl 【V1.08 以降】

正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far tanl(long double x); (C99)
```

[戻り値]

x の正接を返します。
x が非数，または ± ∞である場合，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。

acosh 【V1.08 以降】

双曲線逆余弦 を求めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far acosh(double x); (C99)
```

［戻り値］

x の双曲線逆余弦 を返します。
x が 1 未満の場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。

［詳細説明］

x の双曲線逆余弦 を求めます。角度はラジアン単位で指定します。

acoshf 【V1.08 以降】

双曲線逆余弦 を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far acoshf(float x); (C99)
```

[戻り値]

x の双曲線逆余弦 を返します。
x が 1 未満の場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。

[詳細説明]

x の双曲線逆余弦 を求めます。角度はラジアン単位で指定します。

acoshl 【V1.08 以降】

双曲線逆余弦 を求めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far acoshl(long double x); (C99)
```

［戻り値］

x の双曲線逆余弦 を返します。
x が 1 未満の場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。

［詳細説明］

x の双曲線逆余弦 を求めます。角度はラジアン単位で指定します。

asinh 【V1.08 以降】

双曲線逆正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far asinh(double x); (C99)
```

[戻り値]

x の双曲線逆正弦を返します。

[詳細説明]

x の双曲線逆正弦を求めます。角度はラジアン単位で指定します。

asinhf 【V1.08 以降】

双曲線逆正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far asinhf(float x); (C99)
```

[戻り値]

x の双曲線逆正弦を返します。

[詳細説明]

x の双曲線逆正弦を求めます。角度はラジアン単位で指定します。

asinhl 【V1.08 以降】

双曲線逆正弦を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far asinhl(long double x); (C99)
```

[戻り値]

x の双曲線逆正弦を返します。

[詳細説明]

x の双曲線逆正弦を求めます。角度はラジアン単位で指定します。

atanh 【V1.08 以降】

双曲線逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far atanh(double x); (C99)
```

[戻り値]

x の双曲線逆正接を返します。
x が区間 [-1, +1] 内にない場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 の場合，-HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。
x が 1 の場合，HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x の双曲線逆正接を求めます。角度はラジアン単位で指定します。

atanhf 【V1.08 以降】

双曲線逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far atanhf(float x); (C99)
```

[戻り値]

x の双曲線逆正接を返します。
x が区間 [-1, +1] 内にない場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 の場合，-HUGE_VALF を返し，グローバル変数 errno にはマクロ ERANGE を設定します。
x が 1 の場合，HUGE_VALF を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x の双曲線逆正接を求めます。角度はラジアン単位で指定します。

atanhl 【V1.08 以降】

双曲線逆正接を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far atanhl(long double x); (C99)
```

[戻り値]

x の双曲線逆正接を返します。
x が区間 [-1, +1] 内にない場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 の場合，-HUGE_VALL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。
x が 1 の場合，HUGE_VALL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x の双曲線逆正接を求めます。角度はラジアン単位で指定します。

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 __far cosh(double x);
```

[戻り値]

x の双曲線余弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} + e^{-x}) / 2$$

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 <math.h>
float   __far   coshf(float x);
```

[戻り値]

x の双曲線余弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} + e^{-x}) / 2$$

coshl 【V1.08 以降】

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>
long double __far coshl(long double x); (C99)
```

[戻り値]

x の双曲線余弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} + e^{-x}) / 2$$

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 __far sinh(double x);
```

[戻り値]

x の双曲線正弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} - e^{-x}) / 2$$

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 <math.h>
float   __far   sinhf(float x);
```

[戻り値]

x の双曲線正弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} - e^{-x}) / 2$$

sinhl 【V1.08 以降】

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>
long double __far sinhl(long double x); (C99)
```

[戻り値]

x の双曲線正弦を返します。
オーバフローが生じた場合，∞を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[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。

$$(e^{x} - e^{-x}) / 2$$

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 __far 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*)

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 <math.h>
float   __far   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。

sinhf (*x*) / coshf (*x*)

tanhl 【V1.08 以降】

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>
long double __far tanhl(long double x); (C99)
```

[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。

sinhl (*x*) / coshl (*x*)

exp

指数関数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   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，0，または非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。
オーバフローが生じた場合，∞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]

e の *x* 乗を求めます（e は自然対数の底で，約 2.71828 です）。

expf

指数関数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   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，0，または非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。
オーバフローが生じた場合，∞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]

e の *x* 乗を求めます（e は自然対数の底で，約 2.71828 です）。

expl 【V1.08 以降】

指数関数を求めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far expl(long double x); (C99)
```

［戻り値］

e の *x* 乗を返します。
アンダフローが生じた場合，0，または非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。
オーバフローが生じた場合，∞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］

e の *x* 乗を求めます（e は自然対数の底で，約 2.71828 です）。

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 __far frexp(double val, int *exp);
```

[戻り値]

val の仮数部を返します。返す値は，[1 / 2, 1) の範囲，または 0 です。
val が 0 の場合，**exp* に 0 を設定し，0 を返します。
val が非数，または ± ∞の場合，非数を返し，**exp* に 0 を設定し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

浮動小数点数を正規化した数と 2 の整数べき乗に分割します。整数べき乗を **exp* に格納します。
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 <= | m | < 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。

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 <math.h>
float   __far   frexpf(float val, int *exp);
```

[戻り値]

val の仮数部を返します。返す値は，[1 / 2, 1) の範囲，または 0 です。
val が 0 の場合，**exp* に 0 を設定し，0 を返します。
val が非数，または ± ∞の場合，非数を返し，**exp* に 0 を設定し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

浮動小数点数を正規化した数と 2 の整数べき乗に分割します。整数べき乗を **exp* に格納します。
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 <= | m | < 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。

frexpl 【V1.08 以降】

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>
long double __far frexpl(long double *val*, int **exp*); (C99)

[戻り値]

val の仮数部を返します。返す値は，[1 / 2, 1) の範囲，または 0 です。
val が 0 の場合，**exp* に 0 を設定し，0 を返します。
val が非数，または ± ∞の場合，非数を返し，**exp* に 0 を設定し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

浮動小数点数を正規化した数と 2 の整数べき乗に分割します。整数べき乗を **exp* に格納します。
long 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 <= | m | < 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。

ldexp

浮動小数点数と 2 の整数べき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   ldexp(double val, int exp);
```

[戻り値]

val * 2 ^{exp} で求めた値を返します。
val が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と 2 の整数べき乗を乗算します。

ldexpf

浮動小数点数と 2 の整数べき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   ldexpf(float val, int exp);
```

[戻り値]

val * 2 ^{exp} で求めた値を返します。
val が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と 2 の整数べき乗を乗算します。

ldexpl 【V1.08 以降】

浮動小数点数と 2 の整数べき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

#include <math.h>
long double __far ldexpl(long double *val*, int *exp*); (C99)

[戻り値]

val * 2 ^{exp} で求めた値を返します。
val が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と 2 の整数べき乗を乗算します。

log

自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[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。

logf

自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[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。

logl 【V1.08 以降】

自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far logl(long double x); (C99)
```

[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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[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。

log10

10 を底とした対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far 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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

10 を底とする *x* の対数を求めます。

log10f

10 を底とした対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

10 を底とする *x* の対数を求めます。

log10l 【V1.08 以降】

10 を底とした対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far log10l(long double x); (C99)
```

[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 を設定します。
x が 0 の場合，- ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

10 を底とする *x* の対数を求めます。

log1p 【V1.08 以降】

実引数に 1 を加えた値の自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far log1p(double x); (C99)
```

[戻り値]

x に 1 を加えた値の自然対数を返します。
x が -1 より小さい場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 である場合，-HUGE_VAL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x に 1 を加えた値の自然対数を求めます。

log1pf 【V1.08 以降】

実引数に 1 を加えた値の自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far log1pf(float x); (C99)
```

[戻り値]

x に 1 を加えた値の自然対数を返します。
x が -1 より小さい場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 である場合，-HUGE_VALF を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x に 1 を加えた値の自然対数を求めます。

log1pl 【V1.08 以降】

実引数に 1 を加えた値の自然対数を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far log1pl(long double x); (C99)
```

[戻り値]

x に 1 を加えた値の自然対数を返します。
x が -1 より小さい場合，NaN を返し，グローバル変数 errno にはマクロ EDOM を設定します。
x が -1 である場合，-HUGE_VALL を返し，グローバル変数 errno にはマクロ ERANGE を設定します。

[詳細説明]

x に 1 を加えた値の自然対数を求めます。

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 __far modf(double val, double *iptr);
```

[戻り値]

符号付きの小数部を返します。結果の符号は *val* の符号と同じです。
val が ± ∞の場合，0 を返し，*iptr* に ± ∞，グローバル変数 errno にマクロ EDOM を設定します。
val が非数の場合，非数を返し，*iptr* に非数，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

val を整数部と小数部とに分割し，整数部を **iptr* に格納します。整数部，および小数部の符号は，*val* と同一です。

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 <math.h>
float   __far   modff(float val, float *iptr);
```

[戻り値]

符号付きの小数部を返します。結果の符号は *val* の符号と同じです。
val が ± ∞の場合，0 を返し，*iptr* に ± ∞，グローバル変数 errno にマクロ EDOM を設定します。
val が非数の場合，非数を返し，*iptr* に非数，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

val を整数部と小数部とに分割し，整数部を **iptr* に格納します。整数部，および小数部の符号は，*val* と同一です。

modfl 【V1.08 以降】

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>
long double __far modfl(long double val, long double *iptr); (C99)
```

[戻り値]

符号付きの小数部を返します。結果の符号は *val* の符号と同じです。
val が ± ∞の場合，0 を返し，*iptr* に ± ∞，グローバル変数 errno にマクロ EDOM を設定します。
val が非数の場合，非数を返し，*iptr* に非数，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

val を整数部と小数部とに分割し，整数部を **iptr* に格納します。整数部，および小数部の符号は，*val* と同一です。

scalbn 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far scalbn(double x, int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

scalbnf 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far scalbnf(float x, int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

scalbnl 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far scalbnl(long double x, int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

scalbln 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far scalbln(double x, long int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

scalblnf 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far scalblnf(float x, long int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

scalblnl 【V1.09 以降】

浮動小数点数と FLT_RADIX のべき乗を乗算します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far scalblnl(long double x, long int n); (C99)
```

[戻り値]

x * FLT_RADIX^{n} の値を返します。
x が非数の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
アンダフロー，またはオーバフローが生じた場合，グローバル変数 errno にマクロ ERANGE を設定します。アンダフローの場合，非正規化数を返します。オーバフローの場合，∞を返します。

[詳細説明]

浮動小数点数と FLT_RADIX のべき乗を乗算します。

fabs

絶対値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   fabs(double x);
```

[戻り値]

x の絶対値（大きさ）を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 の絶対値（大きさ）を求めます。

fabsf

絶対値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   fabsf(float x);
```

[戻り値]

x の絶対値（大きさ）を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 の絶対値（大きさ）を求めます。

fabsl 【V1.08 以降】

絶対値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far fabsl(long double x); (C99)
```

[戻り値]

x の絶対値（大きさ）を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 の絶対値（大きさ）を求めます。

pow

x の *y* 乗を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   pow(double x, double y);
```

[戻り値]

x の *y* 乗を返します。
x < 0 で *y* が非整数の場合，または *x* = 0 で *y* <= 0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
オーバフローが発生した場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
アンダフローが発生した場合，非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の *y* 乗を求めます。

powf

x の *y* 乗を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   powf(float x, float y);
```

[戻り値]

x の *y* 乗を返します。
x < 0 で *y* が非整数の場合，または *x* = 0 で *y* <= 0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
オーバフローが発生した場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
アンダフローが発生した場合，非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の *y* 乗を求めます。

powl 【V1.08 以降】

x の *y* 乗を求めます。

[所属]

数学ライブラリ

[指定形式]

#include <math.h>
long double __far powl(long double *x*, long double *y*); (C99)

[戻り値]

x の *y* 乗を返します。
x < 0 で *y* が非整数の場合，または *x* = 0 で *y* <= 0 の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
オーバフローが発生した場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
アンダフローが発生した場合，非正規化数を返し，グローバル変数 errno にマクロ ERANGE を設定します。

[詳細説明]

x の *y* 乗を求めます。

sqrt

平方根を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   sqrt(double x);
```

[戻り値]

x の平方根を返します。
x が負の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

x の平方根を求めます。

sqrtf

平方根を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   sqrtf(float x);
```

[戻り値]

x の平方根を返します。
x が負の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

x の平方根を求めます。

sqrtl 【V1.08 以降】

平方根を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far sqrtl(long double x); (C99)
```

[戻り値]

x の平方根を返します。
x が負の場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。

[詳細説明]

x の平方根を求めます。

ceil

x 以上の最小の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   ceil(double x);
```

[戻り値]

x 以上の最小の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以上の最小の整数値を求めます。

ceilf

x 以上の最小の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   ceilf(float x);
```

[戻り値]

x 以上の最小の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以上の最小の整数値を求めます。

ceill 【V1.08 以降】

x 以上の最小の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far ceill(long double x); (C99)
```

[戻り値]

x 以上の最小の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以上の最小の整数値を求めます。

floor

x 以下の最大の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   floor(double x);
```

[戻り値]

x 以下の最大の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以下の最大の整数値を求めます。

floorf

x 以下の最大の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   floorf(float x);
```

[戻り値]

x 以下の最大の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以下の最大の整数値を求めます。

floorl 【V1.08 以降】

x 以下の最大の整数値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far floorl(long double x); (C99)
```

[戻り値]

x 以下の最大の整数値を返します。
x が ± ∞の場合，± ∞を返し，グローバル変数 errno にマクロ ERANGE を設定します。
x が非数の場合，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 以下の最大の整数値を求めます。

nearbyint 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far nearbyint(double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

nearbyintf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
float __far nearbyintf(float x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

nearbyintl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far nearbyintl(long double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

rint 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far rint(double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。“ 不正確結果 ” 浮動小数点例外は生成しません。

rintf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
float __far rintf(float x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。“ 不正確結果 ” 浮動小数点例外は生成しません。

rintl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far rintl(long double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。“ 不正確結果 ” 浮動小数点例外は生成しません。

lrint 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lrint(double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

lrintf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lrintf(float x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

lrintl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lrintl(long double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

llrint 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llrint(double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

llrintf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llrintf(float x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

llrintl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llrintl(long double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

round 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far round(double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

roundf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
float __far roundf(float x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

roundl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far roundl(long double x); (C99)
```

［戻り値］

丸めた値を返します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

lround 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lround(double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

lroundf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lroundf(float x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

lroundl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long int __far lroundl(long double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

llround 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llround(double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

llroundf 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llroundf(float x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

llroundl 【V1.09 以降】

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long long int __far llroundl(long double x); (C99)
```

［戻り値］

丸めた値を返します。丸めた値が戻り値の型で表現できない場合，0 を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

丸め方向に従って，実引数を浮動小数点形式の整数値に丸めます。実引数がちょうど中間にある場合は，0 から遠い方向を選びます。

trunc 【V1.09 以降】

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far trunc(double x); (C99)
```

［戻り値］

切り捨てた整数値を返します。

［詳細説明］

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

truncf 【V1.09 以降】

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far truncf(float x); (C99)
```

[戻り値]

切り捨てた整数値を返します。

[詳細説明]

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

truncl 【V1.09 以降】

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far truncl(long double x); (C99)
```

[戻り値]

切り捨てた整数値を返します。

[詳細説明]

実引数の小数部を切り捨てた浮動小数点形式の整数値に丸めます。

fmod

剰余を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double  __far   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。
x，または y が非数である場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
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 EDOM を設定します。
x が ± ∞でなく，*y* が ± ∞である場合，*x* 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 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，ある整数 i に対して *x* - i * *y* の値を求めます。その結果は x と同じ符号で，絶対値は *y* の絶対値より小さくなります。

fmodf

剰余を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float   __far   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。
x，または *y* が非数である場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
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 EDOM を設定します。
x が ± ∞でなく，*y* が ± ∞である場合，*x* 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 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，ある整数 i に対して *x* - i * *y* の値を求めます。その結果は x と同じ符号で，絶対値は *y* の絶対値より小さくなります。

fmodl 【V1.08 以降】

剰余を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far fmodl(long double x, long double y); (C99)
```

[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。
x，または *y* が非数である場合，非数を返し，グローバル変数 errno にマクロ EDOM を設定します。
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 EDOM を設定します。
x が ± ∞でなく，*y* が ± ∞である場合，*x* 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 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，ある整数 i に対して *x* - i * *y* の値を求めます。その結果は x と同じ符号で，絶対値は *y* の絶対値より小さくなります。

copysign 【V1.09 以降】

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far copysign(double x, double y); (C99)
```

［戻り値］

x の絶対値をもち，かつ *y* の符号をもつ値を返します。

［詳細説明］

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

copysignf 【V1.09 以降】

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
float __far copysignf(float x, float y); (C99)
```

［戻り値］

x の絶対値をもち，かつ *y* の符号をもつ値を返します。

［詳細説明］

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

copysignl 【V1.09 以降】

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far copysignl(long double x, long double y); (C99)
```

［戻り値］

x の絶対値をもち，かつ *y* の符号をもつ値を返します。

［詳細説明］

x の絶対値をもち，かつ *y* の符号をもつ値を生成します。

nan 【V1.09 以降】

非数を返します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far nan(const char __far *tagp); (C99)
```

［戻り値］

非数を返します。

［詳細説明］

strtod("NAN(n 文字列)", (char __near * __near *)NULL) と等価となります。

nanf 【V1.09 以降】

非数を返します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
float __far nanf(const char __far *tagp); (C99)
```

［戻り値］

非数を返します。

［詳細説明］

strtof("NAN(n 文字列)", (char __near * __near *)NULL) と等価となります。

nanl 【V1.09 以降】

非数を返します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
long double __far nanl(const char __far *tagp); (C99)
```

［戻り値］

非数を返します。

［詳細説明］

strtold("NAN(n 文字列)", (char __near * __near *)NULL) と等価となります。

fdim 【V1.09 以降】

2 個の実引数の正の差を求めます。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
double __far fdim(double x, double y); (C99)
```

［戻り値］

x>y の場合は *x-y* を，*x<=y* の場合は +0 を返します。*x* または *y* が非数の場合は非数を返します。

［詳細説明］

2 個の実引数の正の差を求めます。

fdimf 【V1.09 以降】

2 個の実引数の正の差を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far fdimf(float x, float y); (C99)
```

[戻り値]

x>*y* の場合は *x*-*y* を，*x*<=*y* の場合は +0 を返します。*x* または *y* が非数の場合は非数を返します。

[詳細説明]

2 個の実引数の正の差を求めます。

fdiml 【V1.09 以降】

2 個の実引数の正の差を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far fdiml(long double x, long double y); (C99)
```

[戻り値]

x>*y* の場合は *x*-*y* を，*x*<=*y* の場合は +0 を返します。*x* または *y* が非数の場合は非数を返します。

[詳細説明]

2 個の実引数の正の差を求めます。

fmax 【V1.09 以降】

2 個の引数の大きい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far fmax(double x, double y); (C99)
```

[戻り値]

2 個の実引数の大きい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の大きい方の値を求めます。

fmaxf 【V1.09 以降】

2 個の引数の大きい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far fmaxf(float x, float y); (C99)
```

[戻り値]

2 個の実引数の大きい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の大きい方の値を求めます。

fmaxl 【V1.09 以降】

2 個の引数の大きい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far fmaxl(long double x, long double y); (C99)
```

[戻り値]

2 個の実引数の大きい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の大きい方の値を求めます。

fmin 【V1.09 以降】

2 個の引数の小さい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
double __far fmin(double x, double y); (C99)
```

[戻り値]

2 個の実引数の小さい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の小さい方の値を求めます。

fminf 【V1.09 以降】

2 個の引数の小さい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
float __far fminf(float x, float y); (C99)
```

[戻り値]

2 個の実引数の小さい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の小さい方の値を求めます。

fminl 【V1.09 以降】

2 個の引数の小さい方の値を求めます。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
long double __far fminl(long double x, long double y); (C99)
```

[戻り値]

2 個の実引数の小さい方の値を返します。一方の実引数が非数で，もう一方が非数でない場合は，非数でない方の値を返します。

[詳細説明]

2 個の実引数の小さい方の値を求めます。

isgreater 【V1.09 以降】

最初の実引数が 2 番目の実引数より大きいかどうかを判定します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
int isgreater( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

[戻り値]

(*x*)>(*y*) の値を返します。

[詳細説明]

最初の実引数が 2 番目の実引数より大きいかどうかを判定します。

isgreaterequal 【V1.09 以降】

最初の実引数が 2 番目の実引数より大きい，または等しいかどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int isgreaterequal( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

［戻り値］

(*x*)>=(*y*) の値を返します。

［詳細説明］

最初の実引数が 2 番目の実引数より大きい，または等しいかどうかを判定します。

isless 【V1.09 以降】

最初の実引数が 2 番目の実引数より小さいかどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int isless( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

［戻り値］

(*x*)<(*y*) の値を返します。

［詳細説明］

最初の実引数が 2 番目の実引数より小さいかどうかを判定します。

islessequal 【V1.09 以降】

最初の実引数が 2 番目の実引数より小さい，または等しいかどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int islessequal( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

［戻り値］

(*x*)<=(*y*) の値を返します。

［詳細説明］

最初の実引数が 2 番目の実引数より小さい，または等しいかどうかを判定します。

islessgreater 【V1.09 以降】

最初の実引数が 2 番目の実引数より小さい，または大きいかどうかを判定します。

［所属］

数学ライブラリ

［指定形式］

```
#include <math.h>
int islessgreater( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

［戻り値］

(*x*)>(*y*) || (*x*)<(*y*) の値を返します。

［詳細説明］

最初の実引数が 2 番目の実引数より小さい，または大きいかどうかを判定します。

isunordered 【V1.09 以降】

実引数が順序付けられていないかどうかを判定します。

[所属]

数学ライブラリ

[指定形式]

```
#include <math.h>
int isunordered( 実浮動小数点型 x, 実浮動小数点型 y); (C99)
```

[戻り値]

実引数が順序付けられていない場合は 1 を，そうでない場合は 0 を返します。

[詳細説明]

実引数が順序付けられていないかどうかを判定します。

7.5.5 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 7.7 非局所分岐関数

関数／マクロ名	概要
setjmp	呼び出し環境を保存
longjmp	呼び出し環境を復元

setjmp

呼び出し環境を保存します。

[所属]

標準ライブラリ

[指定形式]

```
#include <setjmp.h>
typedef int    __near  jmp_buf[3];
int    __far   setjmp(jmp_buf env);
```

[戻り値]

setjmp からの戻りの場合 0 を返します。longjmp による非局所分岐の場合，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* に設定します。*env* には，本関数が実行された時点の環境が保存されます。

[注意事項]

setjmp 関数へのポインタを使った間接呼び出しはしないでください。

longjmp

呼び出し環境を復元します。

[所属]

標準ライブラリ

[指定形式]

```
#include <setjmp.h>
typedef int     __near  jmp_buf[3];
void    __far   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。

7.5.6 可変個数実引数関数

可変個数実引数関数として，以下のものがあります。

表 7.8 可変個数実引数関数

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_arg	引数リスト走査用変数の移動
va_copy 【V1.09 以降】	引数リスト走査用変数のコピー（C99）
va_end	引数リスト走査の終了

va_start

引数リスト走査用変数を初期化します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdarg.h>
typedef char    __near  *va_list;
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。

va_arg

引数リスト走査用変数を移動します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdarg.h>
typedef char    __near  *va_list;
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。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。

可変個数引数の場合，既定の実引数拡張に従って型変換されるため，変換後の型を指定してください。実引数がポインタで定数値の場合，実引数にキャストを付けて，ポインタであることを明示してください。既定の実引数拡張については，「(7) 既定の実引数拡張」を参照してください。

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決まります。

va_copy 【V1.09 以降】

引数リスト走査用変数をコピーします。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdarg.h>
typedef char __near *va_list;
void va_copy(va_list dest, va_list src); (C99)
```

［詳細説明］

引数 *dest* を，引数 *src* のコピーとして初期化します。

va_end

引数リスト走査を終了します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdarg.h>
typedef char    __near  *va_list;
void    va_end(va_list ap);
```

[詳細説明]

引数リストの走査を終了します。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。

7.5.7 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
printf	フォーマット指定したテキストを SFR へ書き込み
scanf	フォーマット指定したテキストを SFR から読み込み
snprintf 【V1.07 以降】	フォーマット指定したテキストを配列へ書き込み（C99）
sprintf	フォーマット指定したテキストを配列へ書き込み
sscanf	フォーマット指定したテキストを文字列から読み込み
vprintf	フォーマット指定したテキストを SFR へ書き込み
vscanf 【V1.08 以降】	フォーマット指定したテキストを SFR から読み込み（C99）
vsnprintf 【V1.07 以降】	フォーマット指定したテキストを配列へ書き込み（C99）
vsprintf	フォーマット指定したテキストを配列へ書き込み
vsscanf 【V1.08 以降】	フォーマット指定したテキストを文字列から読み込み（C99）
getchar	SFR からの一文字読み込み
gets	SFR からの文字列読み込み
putchar	SFR への一文字書き込み
puts	文字列からの SFR への書き込み
perror	エラー処理

各関数における stdin からの入力は getchar 関数を介し，stdout への出力は putchar 関数を介す仕様です。
stdin，stdout を変更したい場合は，getchar 関数，putchar 関数を差し替えてください。
perror 関数における stderr への出力は，stdout と同じとし，putchar 関数を介します。putchar 関数を差し替えることで，stderr も変更することになるので注意が必要です。stderr の出力先を stdout と別にしたい場合は，perror 関数を差し替えてください。

printf

フォーマット指定したテキストを SFR へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far printf(const char __far *format, ...);（C90）
int __far printf(const char __far * restrict format, ...);（C99）【V1.07 以降】
int __far printf_tiny(const char __far *format, ...);（C90）
int __far printf_tiny(const char __far * restrict format, ...);（C99）【V1.07 以降】
```

[戻り値]

出力された文字数（null 文字（¥0）は除きます）を返します。
書き込みエラーが発生した時は EOF（-1）を返します。

[詳細説明]

format に続く実引数を出力形式に変換し，putchar 関数を使用して SFR に出力します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。

hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）【V1.07 以降】

h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。

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。

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。

j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）【V1.07 以降】

z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）【V1.07 以降】

t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）【V1.07 以降】

L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。

F 変換は C99 用ライブラリのみ指定可能です。【V1.07 以降】

d，i	int 型の引数を符号付きの 10 進数に変換します。
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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）【V1.07 以降】
e，E	double 型（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

printf_tiny は printf の簡易版です。
-D オプション，または stdio.h のインクルードより前にマクロ __PRINTF_TINY__ を定義した場合は，printf の関数呼び出しを printf_tiny に置き換えます。printf_tiny は変換指定に以下の制限があります。

(1) フラグ
-，+，スペースを指定できません。

(2) フィールド長
負のフィールド長，“*” を指定できません。

(3) 精度
指定できません。

(4) サイズ
ll，j，z，t，L を指定できません。

(5) 型指定文字
f，F，e，E，g，G を指定できません。

［制限］

C99 規格の a，A 変換はサポートしません。

scanf

フォーマット指定したテキストを SFR から読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far  scanf(const char __far *format, ...); (C90)
int __far  scanf(const char __far * restrict format, ...); (C99) 【V1.08 以降】
```

[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]

getchar 関数を使用した SFR からの入力を変換し，*format* に続く実引数が指すオブジェクトに代入します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。指令と矛盾する入力文字によって変換が終了した場合，その矛盾した入力文字は切り捨てられます。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなり，書式内の各指令を順に実行します。入力文字が得られないか，不適切な入力により指令の実行が失敗すると，処理は終了します。

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。 最初の非空白類文字の直前まで (この文字は読まずに残す)，またはそれ以上読み取ることができなくなるまで，入力読み取りを実行します。
通常の文字	“%” 以外のすべての ASCII 文字です。 次の文字を読むことで実行されます。
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
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正の 10 進整数です。0 の場合，規定がないものとします。
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 hh，h，l，ll，j，z，t および L です。

指定がない場合，後ろに続く d，i，o，n，u，x，X の型指定を強制的に int，または unsigned int へのポインタに適用します。さらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を int へのポインタに適用します。

hh 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 signed char，または unsigned char へのポインタに適用します。hh はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を signed char へのポインタに適用します。（C99）【V1.08 以降】

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 int，または unsigned short int へのポインタに適用します。h はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を short int へのポインタに適用します。

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 へのポインタに適用します。l はさらに，後ろに続く f，F，e，E，g，G の型指定を double へのポインタに，n の型指定を long へのポインタに適用します。

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 へのポインタに適用します。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。

j 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 intmax_t，または uintmax_t へのポインタに適用します。j はさらに，後ろに続く n の型指定を intmax_t へのポインタに適用します。（C99）【V1.08 以降】

z 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 size_t，または signed int へのポインタに適用します。z はさらに，後ろに続く n の型指定を signed int へのポインタに適用します。（C99）【V1.08 以降】

t を指定した場合，後ろに続く d，i，o，u，x，X の型指定を ptrdiff_t，または unsigned int へのポインタに適用します。t はさらに，後ろに続く n の型指定を ptrdiff_t へのポインタに適用します。（C99）【V1.08 以降】

L を指定した場合，後ろに続く f，F，e，E，g，G の型指定を強制的に long double へのポインタに適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットです。

(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。

d	10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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。対応する型はサイズ文字に従います。
o	8 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
u	符号なし 10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
x，X	16 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
e，f，g， E，F，G	浮動小数点数，無限大，または NaN を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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 __far *arg*[ ]” にしてください。 ポインタは常に far ポインタとなります。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。

[ ]	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，sscanf 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）。
c	1 文字を走査します。対応する引数は “char __far **arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
p	走査したポインタを格納します。対応する引数は “void __far ***arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
n	入力を読み取りません。これまでに読み取った文字数を，対応する引数に書き込みます。 %n 指令を実行しても，関数終了時に返却する入力項目の個数は増加しません。対応する型はサイズ文字に従います。
%	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。

F 変換は C99 ライブラリのみ指定可能です。

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。

［制限］

a，A 変換および 16 進浮動小数点数はサポートしません。

snprintf 【V1.07 以降】

フォーマット指定したテキストを配列へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far snprintf(char __far * restrict s, size_t n, const char __far * restrict format, ...);（C99）
```

[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。
書き込みエラーが発生した時は EOF（-1）を返します。

[詳細説明]

format に続く実引数を出力形式に変換し，*s* で示された配列に書き込みます。その際の変換方法は，*format* が指す文字列で指定される書式に従います。*n* が 0 の場合，何も書き込まず，*s* はヌル・ポインタでもかまいません。その他の場合，*n*-1 番目より後の出力文字は配列に書き込まずに捨て，配列に実際に書き込んだ文字の列の後にヌル文字を書き込みます。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。
hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）
h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。
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。
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。
j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）
z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）
t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）
L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。
F 変換は C99 用ライブラリのみ指定可能です。

d，i	int 型の引数を符号付きの 10 進数に変換します。
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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）
e，E	double 型（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

［制限］

C99 規格の a，A 変換はサポートしません。

sprintf

フォーマット指定したテキストを配列へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far sprintf(char __far *s, const char __far *format, ...);（C90）
int __far sprintf(char __far * restrict s, const char __far * restrict format, ...);（C99）【V1.07 以降】
int __far sprintf_tiny(char __far *s, const char __far *format, ...);（C90）
int __far sprintf_tiny(char __far * restrict s, const char __far * restrict format, ...);（C99）【V1.07 以降】
```

[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。
書き込みエラーが発生した時は EOF（-1）を返します。

[詳細説明]

format に続く実引数を出力形式に変換し，*s* で示された配列に書き込みます。その際の変換方法は，*format* が指す文字列で指定される書式に従います。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。

hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）【V1.07 以降】

h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。

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。

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。

j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）【V1.07 以降】

z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）【V1.07 以降】

t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）【V1.07 以降】

L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。

F 変換は C99 用ライブラリのみ指定可能です。【V1.07 以降】

d，i	int 型の引数を符号付きの 10 進数に変換します。
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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）【V1.07 以降】
e，E	double 型（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

sprintf_tiny は sprintf の簡易版です。
-D オプション，または stdio.h のインクルードより前にマクロ __PRINTF_TINY__ を定義した場合は，sprintf の関数呼び出しを sprintf_tiny に置き換えます。sprintf_tiny は変換指定に以下の制限があります。

(1) フラグ
-，+，スペースが指定できません。

(2) フィールド長
負のフィールド長，“*” が指定できません。

(3) 精度
指定できません。

(4) サイズ
ll，j，z，t，L を指定できません。

(5) 型指定文字
f，F，e，E，g，G を指定できません。

［制限］

C99 規格の a，A 変換はサポートしません。

sscanf

フォーマット指定したテキストを文字列から読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far  sscanf(const char __far *s, const char __far *format, ...); (C90)
int __far  sscanf(const char __far * restrict s, const char __far * restrict format, ...); (C99) 【V1.08 以降】
```

[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]

s で示された文字列からの入力を変換し，*format* に続く実引数が指すオブジェクトに代入します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなり，書式内の各指令を順に実行します。入力文字が得られないか，不適切な入力により指令の実行が失敗すると，処理は終了します。

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。 最初の非空白類文字の直前まで (この文字は読まずに残す)，またはそれ以上読み取ることができなくなるまで，入力読み取りを実行します。
通常の文字	“%” 以外のすべての ASCII 文字です。 次の文字を読むことで実行されます。
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
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正の 10 進整数です。0 の場合，規定がないものとします。
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 hh，h，l，ll，j，z，t および L です。

指定がない場合，後ろに続く d，i，o，n，u，x，X の型指定を強制的に int，または unsigned int へのポインタに適用します。さらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を int へのポインタに適用します。

hh 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 signed char，または unsigned char へのポインタに適用します。hh はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を signed char へのポインタに適用します。（C99）【V1.08 以降】

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 int，または unsigned short int へのポインタに適用します。h はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を short int へのポインタに適用します。

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 へのポインタに適用します。l はさらに，後ろに続く f，F，e，E，g，G の型指定を double へのポインタに，n の型指定を long へのポインタに適用します。

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 へのポインタに適用します。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。

j 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 intmax_t，または uintmax_t へのポインタに適用します。j はさらに，後ろに続く n の型指定を intmax_t へのポインタに適用します。（C99）【V1.08 以降】

z 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 size_t，または signed int へのポインタに適用します。z はさらに，後ろに続く n の型指定を signed int へのポインタに適用します。（C99）【V1.08 以降】

t を指定した場合，後ろに続く d，i，o，u，x，X の型指定を ptrdiff_t，または unsigned int へのポインタに適用します。t はさらに，後ろに続く n の型指定を ptrdiff_t へのポインタに適用します。（C99）【V1.08 以降】

L を指定した場合，後ろに続く f，F，e，E，g，G の型指定を強制的に long double へのポインタに適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットです。

(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。

d	10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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。対応する型はサイズ文字に従います。
o	8 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
u	符号なし 10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
x，X	16 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
e，f，g， E，F，G	浮動小数点数，無限大，または NaN を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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 __far *arg*[ ]” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。

[ ]	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）。
c	1 文字を走査します。対応する引数は “char __far **arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
p	走査したポインタを格納します。対応する引数は “void __far ***arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
n	入力を読み取りません。これまでに読み取った文字数を，対応する引数に書き込みます。 %n 指令を実行しても，関数終了時に返却する入力項目の個数は増加しません。対応する型はサイズ文字に従います。
%	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。

F 変換は C99 ライブラリのみ指定可能です。

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。

［制限］

a，A 変換および 16 進浮動小数点数はサポートしません。

vprintf

フォーマット指定したテキストを SFR へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far vprintf(const char __far *format, va_list arg);（C90）
int __far vprintf(const char __far * restrict format, va_list arg);（C99）【V1.07 以降】
```

[戻り値]

出力された文字数を返します。

[詳細説明]

引数並びのポインタ *arg* が指す引数を，putchar 関数を使用して SFR に出力します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。

hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）【V1.07 以降】

h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。

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。

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。

j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）【V1.07 以降】

z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）【V1.07 以降】

t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）【V1.07 以降】

L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。

F 変換は C99 用ライブラリのみ指定可能です。【V1.07 以降】

d，i	int 型の引数を符号付きの 10 進数に変換します。
o，u，x，X	unsigned int 型の引数を ddddd 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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）【V1.07 以降】
e，E	double 型（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

［制限］

C99 規格の a，A 変換はサポートしません。

vscanf 【V1.08 以降】

フォーマット指定したテキストを SFR から読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far  vscanf(const char __far * restrict format, va_list arg); (C99) 【V1.08 以降】
```

[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]

getchar 関数を使用した SFR からの入力を変換し，引数並びのポインタ *arg* が指す引数に代入します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。指令と矛盾する入力文字によって変換が終了した場合，その矛盾した入力文字は切り捨てられます。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなり，書式内の各指令を順に実行します。入力文字が得られないか，不適切な入力により指令の実行が失敗すると，処理は終了します。

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。 最初の非空白類文字の直前まで (この文字は読まずに残す)，またはそれ以上読み取ることができなくなるまで，入力読み取りを実行します。
通常の文字	“%” 以外のすべての ASCII 文字です。 次の文字を読むことで実行されます。
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
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正の 10 進整数です。0 の場合，規定がないものとします。
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 hh，h，l，ll，j，z，t および L です。

指定がない場合，後ろに続く d，i，o，n，u，x，X の型指定を強制的に int，または unsigned int へのポインタに適用します。さらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を int へのポインタに適用します。

hh 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 signed char，または unsigned char へのポインタに適用します。hh はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を signed char へのポインタに適用します。（C99）【V1.08 以降】

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 int，または unsigned short int へのポインタに適用します。h はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を short int へのポインタに適用します。

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 へのポインタに適用します。l はさらに，後ろに続く f，F，e，E，g，G の型指定を double へのポインタに，n の型指定を long へのポインタに適用します。

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 へのポインタに適用します。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。

j 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 intmax_t，または uintmax_t へのポインタに適用します。j はさらに，後ろに続く n の型指定を intmax_t へのポインタに適用します。（C99）【V1.08 以降】

z 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 size_t，または signed int へのポインタに適用します。z はさらに，後ろに続く n の型指定を signed int へのポインタに適用します。（C99）【V1.08 以降】

t を指定した場合，後ろに続く d，i，o，u，x，X の型指定を ptrdiff_t，または unsigned int へのポインタに適用します。t はさらに，後ろに続く n の型指定を ptrdiff_t へのポインタに適用します。（C99）【V1.08 以降】

L を指定した場合，後ろに続く f，F，e，E，g，G の型指定を強制的に long double へのポインタに適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットです。

(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。

d	10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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。対応する型はサイズ文字に従います。
o	8 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
u	符号なし 10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
x，X	16 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
e，f，g， E，F，G	浮動小数点数，無限大，または NaN を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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 __far *arg*[ ]” にしてください。 ポインタは常に far ポインタとなります。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。

[ ]	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，sscanf 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）。
c	1 文字を走査します。対応する引数は “char __far *_arg_” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
p	走査したポインタを格納します。対応する引数は “void __far **_arg_” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
n	入力を読み取りません。これまでに読み取った文字数を，対応する引数に書き込みます。 %n 指令を実行しても，関数終了時に返却する入力項目の個数は増加しません。対応する型はサイズ文字に従います。
%	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。

F 変換は C99 ライブラリのみ指定可能です。

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。

［制限］

a，A 変換および 16 進浮動小数点数はサポートしません。

vsnprintf 【V1.07 以降】

フォーマット指定したテキストを配列へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far vsnprintf(char __far * restrict s, size_t n, const char __far * restrict format, va_list arg);（C99）
```

[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。
書き込みエラーが発生した時は EOF（-1）を返します。

[詳細説明]

引数並びのポインタ *arg* が指す引数を，*s* で示された配列に書き込みます。その際の変換方法は，*format* が指す文字列で指定される書式に従います。*n* が 0 の場合，何も書き込まず，*s* はヌル・ポインタでもかまいません。その他の場合，*n*-1 番目より後の出力文字は配列に書き込まずに捨て，配列に実際に書き込んだ文字の列の後にヌル文字を書き込みます。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。

hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）

h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。

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。

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。

j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）

z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）

t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）

L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。

F 変換は C99 用ライブラリのみ指定可能です。

d，i	int 型の引数を符号付きの 10 進数に変換します。
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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）
e，E	double 型（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

［制限］

C99 規格の a，A 変換はサポートしません。

vsprintf

フォーマット指定したテキストを配列へ書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far vsprintf(char __far *s, const char __far *format, va_list arg);（C90）
int __far vsprintf(char __far * restrict s, const char __far * restrict format, va_list arg);（C99）【V1.07 以降】
```

[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。
書き込みエラーが発生した時は EOF（-1）を返します。

[詳細説明]

引数並びのポインタ *arg* が指す引数を，*s* で示された文字列に書き込みます。その際の変換方法は，*format* が指す文字列で指定される書式に従います。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなります。% で始まる変換指定以外は，そのまま出力されます。変換指定は 0 個以上の後続の引数を取り出し，変換して出力されます。

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 “ 代替形式 ” 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，F，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。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，F，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 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，F 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，*s* 変換に対しては最大バイト数です。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 hh，h，l，ll，j，z，t，および L です。

hh 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 signed char，または unsigned char に適用します。hh はさらに，後ろに続く n の型指定を強制的に signed char へのポインタに適用します。（C99）【V1.07 以降】

h 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 short int，または unsigned short int に適用します。h はさらに，後ろに続く n の型指定を強制的に short へのポインタに適用します。

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。

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。

j 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 intmax_t，または uintmax_t に適用します。j はさらに，後ろに続く n の型指定を強制的に intmax_t へのポインタに適用します。（C99）【V1.07 以降】

z 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 size_t，または signed int に適用します。z はさらに，後ろに続く n の型指定を強制的に signed int へのポインタに適用します。（C99）【V1.07 以降】

t 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 ptrdiff_t，または unsigned int に適用します。t はさらに，後ろに続く n の型指定を強制的に ptrdiff_t へのポインタに適用します。（C99）【V1.07 以降】

L を指定した場合，後ろに続く e，E，f，F，g，G の型指定を強制的に long double に適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットであるため，指定による影響はありません。

(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。

F 変換は C99 用ライブラリのみ指定可能です。【V1.07 以降】

d，i	int 型の引数を符号付きの 10 進数に変換します。
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。

f，F	double 型（単精度用の関数では float 型）の引数を [-]dddd.dddd の形式の 10 進表記に変換します。 無限大を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]inf，F 変換に対しては [-]INF を用います。NaN を表す double 型の引数を変換したときの形式は，f 変換に対しては [-]nan，F 変換に対しては [-]NAN を用います。（C99）【V1.07 以降】
e，E	double 型（単精度用の関数では float 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.ddde±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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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（単精度用の関数では float 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。 無限大または NaN を表す double 型の引数を変換したときの形式は，f または F 変換指定子と同じです。（C99）【V1.07 以降】
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。
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。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。
p	ポインタの値を出力します。ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
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。
%	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。

［制限］

C99 規格の a，A 変換はサポートしません。

vsscanf 【V1.08 以降】

フォーマット指定したテキストを文字列から読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far  vsscanf(const char __far * restrict s, const char __far * restrict format, va_list arg); (C99)  【V1.08 以降】
```

[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]

s で示された文字列からの入力を変換し，引数並びのポインタ *arg* が指す引数に代入します。その際の変換方法は，*format* が指す文字列で指定される書式に従います。領域の重なり合うオブジェクト間で複写が行われるとき，動作は保証されません。

書式に対して実引数が不足しているときの動作は保証されません。実引数が残っているにもかかわらず，書式が尽きてしまう場合，余分な実引数は評価するだけで無視されます。
書式は 0 個以上の指令からなり，書式内の各指令を順に実行します。入力文字が得られないか，不適切な入力により指令の実行が失敗すると，処理は終了します。

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。 最初の非空白類文字の直前まで (この文字は読まずに残す)，またはそれ以上読み取ることができなくなるまで，入力読み取りを実行します。
通常の文字	“%” 以外のすべての ASCII 文字です。 次の文字を読むことで実行されます。
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
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正の 10 進整数です。0 の場合，規定がないものとします。
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 hh，h，l，ll，j，z，t および L です。

指定がない場合，後ろに続く d，i，o，n，u，x，X の型指定を強制的に int，または unsigned int へのポインタに適用します。さらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を int へのポインタに適用します。

hh 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 signed char，または unsigned char へのポインタに適用します。hh はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を signed char へのポインタに適用します。（C99）【V1.08 以降】

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 int，または unsigned short int へのポインタに適用します。h はさらに，後ろに続く f，F，e，E，g，G の型指定を float へのポインタに，n の型指定を short int へのポインタに適用します。

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 へのポインタに適用します。l はさらに，後ろに続く f，F，e，E，g，G の型指定を double へのポインタに，n の型指定を long へのポインタに適用します。

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 へのポインタに適用します。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。

j 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 intmax_t，または uintmax_t へのポインタに適用します。j はさらに，後ろに続く n の型指定を intmax_t へのポインタに適用します。（C99）【V1.08 以降】

z 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 size_t，または signed int へのポインタに適用します。z はさらに，後ろに続く n の型指定を signed int へのポインタに適用します。（C99）【V1.08 以降】

t を指定した場合，後ろに続く d，i，o，u，x，X の型指定を ptrdiff_t，または unsigned int へのポインタに適用します。t はさらに，後ろに続く n の型指定を ptrdiff_t へのポインタに適用します。（C99）【V1.08 以降】

L を指定した場合，後ろに続く f，F，e，E，g，G の型指定を強制的に long double へのポインタに適用します。ただし，本コンパイラでは double 型と long double 型は同じフォーマットです。

(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。

d	10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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。対応する型はサイズ文字に従います。
o	8 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
u	符号なし 10 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
x，X	16 進整数を対応する引数に読み込みます。対応する型はサイズ文字に従います。
e，f，g， E，F，G	浮動小数点数，無限大，または NaN を対応する引数に読み込みます。対応する型はサイズ文字に従います。
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 __far *arg*[ ]” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。null ポインタを渡した場合の動作は保証されません。

[ ]	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）。
c	1 文字を走査します。対応する引数は “char __far **arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
p	走査したポインタを格納します。対応する引数は “void __far ***arg*” にしてください。 ポインタは常に far ポインタとしてください。定数を渡す場合は，実引数にキャストをつけてポインタであることを明示してください。
n	入力を読み取りません。これまでに読み取った文字数を，対応する引数に書き込みます。 %n 指令を実行しても，関数終了時に返却する入力項目の個数は増加しません。対応する型はサイズ文字に従います。
%	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。

F 変換は C99 ライブラリのみ指定可能です。

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。

［制限］

a，A 変換および 16 進浮動小数点数はサポートしません。

getchar

SFR から文字を読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far   getchar(void);
```

[戻り値]

SFR から読み込んだ値を返します。

[詳細説明]

SFR である P0 から 1 文字を読み取ります。読み込みに関してエラー・チェックは行いません。

[注意事項]

- stdin を変更したい場合は，本関数を差し替えてください。

gets

SFR から文字列を読み込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
char    __near  *__far  gets(char __near *s);
char    __far   *__far  _COM_gets_f(char __far *s);
```

[戻り値]

s を返します。
ファイルの終わりを検出し，かつ配列に 1 文字も読み取っていなかった場合，配列の内容を変化させずに残し，null ポインタを返します。

[詳細説明]

getchar 関数を使用して SFR から文字列を読み取り，*s* が指す配列に格納します。
ファイルの終わりを検出した時，または改行文字を読み取った時に，文字の読み取りを終了し，読み取った改行文字を捨て，最後に配列に格納した文字の直後に null 文字を書きます。

putchar

SFR へ文字を書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int __far   putchar(int c);
```

[戻り値]

文字 *c* を返します。

[詳細説明]

SFR である P0 へ文字 c を書き込みます。書き込みに関してエラー・チェックは行いません。

[注意事項]

- stdout を変更したい場合は，本関数を差し替えてください。なお，本関数を差し替えることで，stderr も合わせて変更となるので注意が必要です。stderr の出力先を stdout と別にしたい場合は，perror 関数を変更してください。

puts

SFR へ文字列を書き込みます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
int     __far  puts(const char __near *s);
int     __far   _COM_puts_f(const char __far *s);
```

[戻り値]

0 を返します。
putchar 関数が -1 を返した場合は -1 を返します。

[詳細説明]

putchar 関数を使用して SFR へ文字列 s を書き込みます。文字列の終端 null 文字は書き込まず，代わりに改行文字を書き込みます。

perror

エラー・メッセージを生成します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdio.h>
void    __far   perror(const char __near *s);
void    __far   _COM_perror_f(const char __far *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。
stderr は stdout と同じで，SFR である P0 とします。putchar 関数を使用して SFR に出力します。
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	`printf("%s:%s\n", s, s_fix);`
s が NULL の場合	`printf("%s\n", s_fix);`

s_fix は，次のようになります。

errno が 0 の場合	"No error"
errno が EDOM の場合	"EDOM error"
errno が ERANGE の場合	"ERANGE error"
その他の場合	"Unknown error"

[注意事項]

- putchar 関数を差し替えることで，stderr も合わせて変更となるので注意が必要です。stderr の出力先を stdout と別にしたい場合は，本関数を変更してください。

7.5.8 一般ユーティリティ関数

一般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 一般ユーティリティ関数

関数／マクロ名	概要
atof	文字列を浮動小数点数（double 型）へ変換
atoff	文字列を浮動小数点数（float 型）へ変換
atoi	文字列を整数（int 型）へ変換
atol	文字列を整数（long int 型）へ変換
atoll 【V1.07 以降】	文字列を整数（long long int 型）へ変換（C99）
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）
strtof	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）
strtold 【V1.07 以降】	文字列を浮動小数点数（long double 型）へ変換（最終文字列へのポインタ格納）（C99）
strtol	文字列を整数（long int 型）へ変換し，最終文字列へのポインタを格納
strtoll 【V1.07 以降】	文字列を整数（long long int 型）へ変換し，最終文字列へのポインタを格納（C99）
strtoul	文字列を整数（unsigned long int 型）へ変換し，最終文字列へのポインタを格納
strtoull 【V1.07 以降】	文字列を整数（unsigned long long int 型）へ変換し，最終文字列へのポインタを格納（C99）
rand	疑似乱数列生成
srand	疑似乱数列の種類を設定
calloc 【V1.02 以降】	0 初期化される動的メモリの割り当て
free 【V1.02 以降】	動的メモリの解放
malloc 【V1.02 以降】	動的メモリの割り当て
realloc 【V1.02 以降】	動的メモリの再割り当て
abort	プログラムを異常終了する
bsearch	バイナリ検索
qsort	整列
abs	絶対値（int 型）を出力
div	除算（int 型）
labs	絶対値（long 型）を出力
ldiv	除算（long 型）
llabs 【V1.07 以降】	絶対値（long long 型）を出力（C99）
lldiv 【V1.07 以降】	除算（long long 型）（C99）

atof

文字列を浮動小数点数（double 型）へ変換します。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
double  __far   atof(const char __near *nptr);
double  __far   _COM_atof_f(const char __far *nptr);
```

［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。
オーバフローが生じる場合，±∞を返し，マクロ ERANGE をグローバル変数 errno に設定します。
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 に設定します。

［詳細説明］

nptr の指す文字列の最初の部分を double 型の表現に変換します。

atoff

文字列を浮動小数点数（float 型）へ変換します。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
float   __far   atoff(const char __near *nptr);
float   __far   _COM_atoff_f(const char __far *nptr);
```

［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。
オーバフローが生じる場合，±∞を返し，マクロ ERANGE をグローバル変数 errno に設定します。
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 に設定します。

［詳細説明］

nptr の指す文字列の最初の部分を float 型の表現に変換します。

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 __far   atoi(const char __near *nptr);
int __far   _COM_atoi_f(const char __far *nptr);
```

［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。
オーバフローが生じる場合，正なら INT_MAX，負なら INT_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の最初の部分を int 型の 10 進数表現に変換します。

atol

文字列を整数（long int 型）へ変換します。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
long int    __far   atol(const char __near *nptr);
long int    __far   _COM_atol_f(const char __farr *nptr);
```

［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。
オーバフローが生じる場合，正なら LONG_MAX，負なら LONG_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の最初の部分を long int 型の 10 進数表現に変換します。

atoll 【V1.07 以降】

文字列を整数（long long int 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 int __far atoll(const char __near *nptr);（C99）
long long int __far _COM_atoll_f(const char __far *nptr);（C99）
```

［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。
オーバフローが生じる場合，正なら LLONG_MAX，負なら LLONG_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の最初の部分を long long int 型の 10 進数表現に変換します。

strtod

文字列を浮動小数点数（double 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>
double __far strtod(const char __near *nptr, char __near * __near *endptr);（C90）
double __far strtod(const char __near * restrict nptr, char __near * __near * restrict endptr);（C99）【V1.07 以降】
double __far _COM_strtod_ff(const char __far *nptr, char __far * __far *endptr);（C90）
double __far _COM_strtod_ff(const char __far * restrict nptr, char __far * __far * restrict endptr);（C99）【V1.07 以降】
```

［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。
オーバフローが生じる場合，±∞を返し，マクロ ERANGE をグローバル変数 errno に設定します。
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 に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を double 型の表現に変換します。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

［制限］

C99 規格の変換対象列の認識可能な形式で，16 進浮動小数点数はサポートしません。

strtof

文字列を浮動小数点数（float 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>
float __far strtof(const char __near *nptr, char __near * __near *endptr);（C90）
float __far strtof(const char __near * restrict nptr, char __near * __near * restrict endptr);（C99）【V1.07 以降】
float __far _COM_strtof_ff(const char __far *nptr, char __far * __far *endptr);（C90）
float __far _COM_strtof_ff(const char __far * restrict nptr, char __far * __far * restrict endptr);（C99）【V1.07 以降】
```

[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。
オーバフローが生じる場合，±∞を返し，マクロ ERANGE をグローバル変数 errno に設定します。
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 に設定します。

[詳細説明]

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を float 型の表現に変換します。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

[制限]

C99 規格の変換対象列の認識可能な形式で，16 進浮動小数点数はサポートしません。

strtold 【V1.07 以降】

文字列を浮動小数点数（long double 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 double __far strtold(const char __near * restrict nptr, char __near * __near * restrict endptr);（C99）
long double __far _COM_strtold_ff(const char __far * restrict nptr, char __far * __far * restrict endptr);（C99）
```

［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。
オーバフローが生じる場合，±∞を返し，マクロ ERANGE をグローバル変数 errno に設定します。
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 に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を long double 型の表現に変換します。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

［制限］

C99 規格の変換対象列の認識可能な形式で，16 進浮動小数点数はサポートしません。

strtol

文字列を整数（long int 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 int __far strtol(const char __near *nptr, char __near * __near *endptr, int base);（C90）
long int __far strtol(const char __near * restrict nptr, char __near * __near * restrict endptr, int base);（C99）【V1.07 以降】
long int __far _COM_strtol_ff(const char __far *nptr, char __far * __far *endptr, int base);（C90）
long int __far _COM_strtol_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base);（C99）【V1.07 以降】
```

［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。
オーバフローが生じる場合，LONG_MAX，または LONG_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を long int 型の表現に変換します。base が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

strtoll 【V1.07 以降】

文字列を整数（long long int 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 int __far strtoll(const char __near * restrict nptr, char __near * __near * restrict endptr, int base);（C99）
long long int __far _COM_strtoll_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base);（C99）
```

［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。
オーバフローが生じる場合，LLONG_MAX，または LLONG_MIN を返し，マクロ ERANGE をグローバル変数 errno に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を long long int 型の表現に変換します。*base* が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

strtoul

文字列を整数（unsigned long int 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 int __far strtoul(const char __near *nptr, char __near * __near *endptr, int base);（C90）
unsigned long int __far strtoul(const char __near * restrict nptr, char __near * __near * restrict endptr, int base);（C99）【V1.07 以降】
unsigned long int __far _COM_strtoul_ff(const char __far *nptr, char __far * __far *endptr, int base);（C90）
unsigned long int __far _COM_strtoul_ff(const char __far * restrict nptr, char __far * __far * restrict endptr, int base);（C99）【V1.07 以降】
```

［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 に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を unsigned long int 型の表現に変換します。base が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

strtoull 【V1.07 以降】

文字列を整数（unsigned long long int 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 int __far strtoull(const char __near * restrict *nptr*, char __near * __near * restrict *endptr*, int *base*);（C99）
unsigned long long int __far _COM_strtoull_ff(const char __far * restrict *nptr*, char __far * __far * restrict *endptr*, int *base*);（C99）

［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 に設定します。

［詳細説明］

nptr の指す文字列の先頭から 0 個以上の空白類文字（isspace 関数が真となる文字）の列を読み飛ばし，次の文字からの文字列を unsigned long long int 型の表現に変換します。*base* が 0 である場合は C の基数表現と解釈し，*base* が 2 以上 36 以下である場合はその値を基数とします。*endptr* が null ポインタでない場合は変換されなかった残りの文字列へのポインタを *endptr* に設定します。

rand

疑似乱数を作ります。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
int __far   rand(void);
```

[戻り値]

乱数を返します。

[詳細説明]

0 以上 RAND_MAX（0x7FFF）以下の乱数を返します。

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 __far  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。

calloc 【V1.02 以降】

ゼロ初期化したメモリを割り当てます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void __near * __far 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。
nmemb または *size* が 0 の場合，あるいは領域を割り付けられない場合は 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，その領域を 0 で初期化します。

【Professional 版のみ】【V1.03 以降】
セキュリティ機能用 malloc 系ライブラリを使用する場合，次の操作を行った場合に __heap_chk_fail 関数を呼び出します。

- calloc，malloc，realloc で割り付けた領域以外のポインタを free，realloc に渡す。
- free で開放した後のポインタを再度 free，realloc に渡す。
- calloc，malloc，realloc で割り当てた領域の外側（前後各 2 バイト）に何らかの値を書き込んだ後，割り当てた領域を指すポインタを free，realloc に渡す。

__heap_chk_fail 関数はユーザが定義する必要があり，動的メモリ管理の異常時に実行する処理を記述します。
__heap_chk_fail 関数を定義する際には，次の項目に注意してください。

- __heap_chk_fail 関数は，返却値および引数の型が void 型の far 関数とします。
  void __far __heap_chk_fail(void);
- __heap_chk_fail 関数は，static 関数にしないでください。
- __heap_chk_fail 関数内で再帰的にヒープ・メモリ領域の破壊を検出しないでください。

セキュリティ機能用の calloc，malloc，および realloc は，領域外への書き込みを検出するために前後各 2 バイト余分に領域を割り当てます。このため，通常より多くヒープ・メモリ領域を消費します。

[注意事項]

ヒープ・メモリ領域のデフォルト・サイズは 0x100 バイトです。
ヒープ・メモリ領域を変更する場合は，配列 _REL_sysheap を定義し，配列のサイズを変数 _REL_sizeof_sysheap に設定してください。

【ヒープ・メモリ領域設定例】
```
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考　　配列 _REL_sysheap は，偶数番地に配置してください。

free 【V1.02 以降】

メモリを解放します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void __far free(void __near *ptr);
```

[詳細説明]

ptr が指す領域を解放します。*ptr* が null ポインタである場合は何もしません。それ以外で，*ptr* が calloc，malloc，および realloc で割り付けた領域でない場合，または，*ptr* が free および realloc によって既に解放されていた場合，動作を保証しません。

【Professional 版のみ】【V1.03 以降】
セキュリティ機能用 malloc 系ライブラリを使用する場合，次の操作を行った場合に __heap_chk_fail 関数を呼び出します。

- calloc，malloc，realloc で割り付けた領域以外のポインタを free，realloc に渡す。
- free で開放した後のポインタを再度 free，realloc に渡す。
- calloc，malloc，realloc で割り当てた領域の外側（前後各 2 バイト）に何らかの値を書き込んだ後，割り当てた領域を指すポインタを free，realloc に渡す。

__heap_chk_fail 関数はユーザが定義する必要があり，動的メモリ管理の異常時に実行する処理を記述します。
__heap_chk_fail 関数を定義する際には，次の項目に注意してください。

- __heap_chk_fail 関数は，返却値および引数の型が void 型の far 関数とします。
  void __far __heap_chk_fail(void);
- __heap_chk_fail 関数は，static 関数にしないでください。
- __heap_chk_fail 関数内で再帰的にヒープ・メモリ領域の破壊を検出しないでください。

セキュリティ機能用の calloc，malloc，および realloc は，領域外への書き込みを検出するために前後各 2 バイト余分に領域を割り当てます。このため，通常より多くヒープ・メモリ領域を消費します。

malloc 【V1.02 以降】

メモリを割り当てます。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void __near * __far 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。
size が 0 の場合，あるいは領域を割り付けられない場合は null ポインタを返します。

[詳細説明]

大きさが *size* の領域を割り当てます。領域は初期化されません。

【Professional 版のみ】【V1.03 以降】
セキュリティ機能用 malloc 系ライブラリを使用する場合，次の操作を行った場合に __heap_chk_fail 関数を呼び出します。

- calloc，malloc，realloc で割り付けた領域以外のポインタを free，realloc に渡す。
- free で開放した後のポインタを再度 free，realloc に渡す。
- calloc，malloc，realloc で割り当てた領域の外側（前後各 2 バイト）に何らかの値を書き込んだ後，割り当てた領域を指すポインタを free，realloc に渡す。

__heap_chk_fail 関数はユーザが定義する必要があり，動的メモリ管理の異常時に実行する処理を記述します。
__heap_chk_fail 関数を定義する際には，次の項目に注意してください。

- __heap_chk_fail 関数は，返却値および引数の型が void 型の far 関数とします。
  void __far __heap_chk_fail(void);
- __heap_chk_fail 関数は，static 関数にしないでください。
- __heap_chk_fail 関数内で再帰的にヒープ・メモリ領域の破壊を検出しないでください。

セキュリティ機能用の calloc，malloc，および realloc は，領域外への書き込みを検出するために前後各 2 バイト余分に領域を割り当てます。このため，通常より多くヒープ・メモリ領域を消費します。

[注意事項]

ヒープ・メモリ領域のデフォルト・サイズは 0x100 バイトです。
ヒープ・メモリ領域を変更する場合は，配列 _REL_sysheap を定義し，配列のサイズを変数 _REL_sizeof_sysheap に設定してください。

【ヒープ・メモリ領域設定例】
```
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考　　配列 _REL_sysheap は，偶数番地に配置してください。

realloc 【V1.02 以降】

メモリを再割り当てします。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void __near * __far realloc(void __near *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。
size が 0 の場合，あるいは領域を割り付けられない場合は null ポインタを返します。

[詳細説明]

ptr が指す領域を *size* が示す大きさに変更します。
再割り当て前の大きさと，*size* の小さい方までの領域の内容は変わりません。
大きさが増加する場合，増加分の領域は初期化されません。
ptr が null ポインタの場合は，“malloc（*size*）” と同じ動作をします。
ptr が null ポインタ以外で *size* が 0 の場合は，“free（*ptr*）” と同じ動作をします。
それ以外で，*ptr* が calloc，malloc，および realloc で割り付けた領域でない場合，または，*ptr* が free および realloc によって既に解放されていた場合，動作を保証しません。

【Professional 版のみ】【V1.03 以降】
セキュリティ機能用 malloc 系ライブラリを使用する場合，次の操作を行った場合に __heap_chk_fail 関数を呼び出します。

- calloc，malloc，realloc で割り付けた領域以外のポインタを free，realloc に渡す。
- free で開放した後のポインタを再度 free，realloc に渡す。
- calloc，malloc，realloc で割り当てた領域の外側（前後各 2 バイト）に何らかの値を書き込んだ後，割り当てた領域を指すポインタを free，realloc に渡す。

__heap_chk_fail 関数はユーザが定義する必要があり，動的メモリ管理の異常時に実行する処理を記述します。
__heap_chk_fail 関数を定義する際には，次の項目に注意してください。

- __heap_chk_fail 関数は，返却値および引数の型が void 型の far 関数とします。
  void __far __heap_chk_fail(void);
- __heap_chk_fail 関数は，static 関数にしないでください。
- __heap_chk_fail 関数内で再帰的にヒープ・メモリ領域の破壊を検出しないでください。

セキュリティ機能用の calloc，malloc，および realloc は，領域外への書き込みを検出するために前後各 2 バイト余分に領域を割り当てます。このため，通常より多くヒープ・メモリ領域を消費します。

[注意事項]

ヒープ・メモリ領域のデフォルト・サイズは 0x100 バイトです。
ヒープ・メモリ領域を変更する場合は，配列 _REL_sysheap を定義し，配列のサイズを変数 _REL_sizeof_sysheap に設定してください。

【ヒープ・メモリ領域設定例】

```
#include <stddef.h>
#define SIZEOF_HEAP  0x200
char _REL_sysheap[SIZEOF_HEAP];
size_t _REL_sizeof_sysheap = SIZEOF_HEAP;
```

備考　　配列 _REL_sysheap は，偶数番地に配置してください。

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 __far  abort(void);
```

［詳細説明］

プログラムを異常終了します。呼び出し元には復帰しません。

bsearch

整列済みの配列から値を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <stdlib.h>
void __near * __far bsearch(const void __near *key, const void __near *base, size_t nmemb, size_t size, int (__far *compar)(const void __near *, const void __near *));
void __far * __far  _COM_bsearch_f(const void __far *key, const void __far *base, size_t nmemb, size_t size, int (__far *compar)(const void __far *, const void __far *));
```

[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。

compar が指す比較関数は，key オブジェクトへのポインタを第 1 引数とし，配列要素へのポインタを第 2 引数として呼び出されます。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します。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。

qsort

配列を整列します。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
void __far  qsort(void __near *base, size_t nmemb, size_t size, int (__far *compar)(const void __near *, const void __near *));
void __far  _COM_qsort_f(void __far *base, size_t nmemb, size_t size, int (__far *compar)(const void __far *, const void __far *));
```

［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。

2 つの要素が等しいとき，整列された配列内でのそれらの順序は保証されません。

abs

絶対値（int 型）を求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
int __far   abs(int j);
```

［戻り値］

j の絶対値（*j* の大きさ），| *j* | を返します。abs の入力値が負の最小値である場合は，同値を返します。

［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。

div

int 型の除算を行い，商と余りを求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
div_t __far div(int numer, int denom);
```

［戻り値］

除算の結果を格納した構造体を返します。0 で割った場合，商 quot は -1，剰余 rem は *numer* が設定されます。

［詳細説明］

int 型の値を除算する場合に使用します。
分子 *numer* を分母 *denom* で割ったその商 quot と剰余 rem 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;
```

割り切れない場合，結果の商は，代数的な商に最も近くそれより絶対値が小さい整数となります。

labs

絶対値（long 型）を求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
long int __far  labs(long int j);
```

［戻り値］

j の絶対値（*j* の大きさ），| *j* | を返します。labs の入力値が負の最小値である場合は，同値を返します。

［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。

ldiv

long 型の除算を行い，商と余りを求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
ldiv_t __far    ldiv(long int numer, long int denom);
```

［戻り値］

除算の結果を格納した構造体を返します。0 で割った場合，商 quot は -1，剰余 rem は *numer* が設定されます。

［詳細説明］

long 型の値を除算する場合に使用します。
分子 *numer* を分母 *denom* で割ったその商 quot と剰余 rem 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 int    quot;
    long int    rem;
} ldiv_t;
```

割り切れない場合，結果の商は，代数的な商に最も近くそれより絶対値が小さい整数となります。

llabs 【V1.07 以降】

絶対値（long long 型）を求めます。

［所属］

標準ライブラリ

［指定形式］

```
#include <stdlib.h>
long long int __far llabs(long long int j);（C99）
```

［戻り値］

j の絶対値（*j* の大きさ），|*j*| を返します。llabs の入力値が負の最小値である場合は，同値を返します。

［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。

lldiv 【V1.07 以降】

long long 型の除算を行い，商と余りを求めます。

［所属］

標準ライブラリ

［指定形式］

#include <stdlib.h>
lldiv_t __far lldiv(long long int *numer*, long long int *denom*);（C99）

［戻り値］

除算の結果を格納した構造体を返します。0 で割った場合，商 quot は -1，剰余 rem は *numer* が設定されます。

［詳細説明］

long long 型の値を除算する場合に使用します。
分子 *numer* を分母 *denom* で割ったその商 quot と剰余 rem 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 int quot;
    long long int rem;
} lldiv_t;
```

割り切れない場合，結果の商は，代数的な商に最も近くそれより絶対値が小さい整数となります。

7.5.9 文字列操作関数

文字列操作関数として，以下のものがあります。

表 7.11 文字列操作関数

関数／マクロ名	概要
memcpy	メモリ・コピー
memmove	メモリ移動
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）
memcmp	メモリ比較
strcmp	文字列比較
strncmp	文字列比較（文字数指定）
memchr	メモリ検索
strchr	文字列検索（指定文字の最初の位置）
strcspn	文字列検索（指定文字列を含まない最大の長さ）
strpbrk	文字列検索（指定したいずれかの文字）
strrchr	文字列検索（最後の位置）
strspn	文字列検索（指定文字列を含む最大の長さ）
strstr	文字列検索（指定文字列の最初の位置）
strtok	トークン分割
memset	オブジェクトの初期化
strerror	エラー・メッセージの取得
strlen	文字列の長さ

memcpy

オブジェクトをコピーします。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void __near * __far memcpy(void __near *s1, const void __near *s2, size_t n);（C90）
void __near * __far memcpy(void __near * restrict s1, const void __near * restrict s2, size_t n);（C99）【V1.07 以降】
void __far * __far _COM_memcpy_ff(void __far *s1, const void __far *s2, size_t n);（C90）
void __far * __far _COM_memcpy_ff(void __far * restrict s1, const void __far * restrict s2, size_t n);（C99）【V1.07 以降】
```

[戻り値]

s1 の値を返します。

[詳細説明]

n 文字分を *s2* の指すオブジェクトから *s1* の指すオブジェクトへコピーします。
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。

[注意事項]

memcpy 関数は CC-RL が内部的に呼び出すことがあり，ランタイム・ライブラリに含まれます。同名のユーザ関数を作成しないでください。

memmove

オブジェクトをコピーします。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void __near * __far memmove(void __near *s1, void __near *s2, size_t n);
void __far * __far  _COM_memmove_ff(void __far *s1, void __far *s2, size_t n);
```

[戻り値]

s1 の値を返します。

[詳細説明]

n 個の文字を，*s2* の指すオブジェクトから *s1* の指すオブジェクトへコピーします。
コピー元とコピー先の領域が重なっている場合も正しくコピーします。

strcpy

文字列をコピーします。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strcpy(char __near *s1, const char __near *s2);（C90）
char __near * __far strcpy(char __near * restrict s1, const char __near * restrict s2);（C99）【V1.07 以降】
char __far * __far _COM_strcpy_ff(char __far *s1, const char __far *s2);（C90）
char __far * __far _COM_strcpy_ff(char __far * restrict s1, const char __far * restrict s2);（C99）【V1.07 以降】
```

[戻り値]

s1 の値を返します。

[詳細説明]

s2 の指す文字列（終端の null 文字を含む）を *s1* の指す配列にコピーします。
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。

strncpy

文字数を指定文字数分コピーします。

[所属]

標準ライブラリ

[指定形式]

#include <string.h>
char __near * __far strncpy(char __near *s1, const char __near *s2, size_t n);（C90）
char __near * __far strncpy(char __near * restrict s1, const char __near * restrict s2, size_t n);（C99）【V1.07 以降】
char __far * __far _COM_strncpy_ff(char __far *s1, const char __far *s2, size_t n);（C90）
char __far * __far _COM_strncpy_ff(char __far * restrict s1, const char __far * restrict s2, size_t n);（C99）【V1.07 以降】

[戻り値]

s1 の値を返します。

[詳細説明]

s2 の指す配列から *s1* の指す配列に最大で *n* 文字（null 文字，および null 文字に続く文字列は付加しない）コピーします。*s2* の指す配列が *n* 文字より短い文字列の場合，全部で *n* 文字分書き込まれるまで，*s1* の指す配列内のコピーに null 文字（¥0）が付加されます。s2 が指す配列が n 文字以上である場合，null 文字は付加されません。
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。

strcat

文字列を連結します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strcat(char __near *s1, const char __near *s2);（C90）
char __near * __far strcat(char __near * restrict s1, const char __near * restrict s2);（C99）【V1.07 以降】
char __far * __far _COM_strcat_ff(char __far *s1, const char __far *s2);（C90）
char __far * __far _COM_strcat_ff(char __far * restrict s1, const char __far * restrict s2);（C99）【V1.07 以降】
```

[戻り値]

s1 の値を返します。

[詳細説明]

s2 の指す文字列のコピーを，null 文字（¥0）を含めて，*s1* の指す文字列の末尾に連結します。*s2* の最初の文字は *s1* の終わりの null 文字（¥0）を上書きします。
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。

strncat

文字数を指定文字数分連結します。

[所属]

標準ライブラリ

[指定形式]

#include <string.h>
char __near * __far strncat(char __near *s1, const char __near *s2, size_t n);（C90）
char __near * __far strncat(char __near * restrict s1, const char __near * restrict s2, size_t n);（C99）【V1.07 以降】
char __far * __far _COM_strncat_ff(char __far *s1, const char __far *s2, size_t n);（C90）
char __far * __far _COM_strncat_ff(char __far * restrict s1, const char __far * restrict s2, size_t n);（C99）【V1.07 以降】

[戻り値]

s1 の値を返します。

[詳細説明]

s2 の指す配列の先頭から，最大で *n* 文字（null 文字，および null 文字に続く文字列は付加しない）を *s1* の指す文字列の末尾に連結します。*s2* の最初の文字は *s1* 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。
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。

[注意事項]

null 文字（¥0）は常に付加されるので，コピーが *n* によって制限される場合，*s1* に付加される文字の個数は *n* + 1 になることに注意してください。

memcmp

オブジェクトを比較します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
int __far   memcmp(const void __near *s1, const void __near *s2, size_t n);
int __far   _COM_memcmp_ff(const void __far *s1, const void __far *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 より小さい値を返します。

[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。

strcmp

文字列を比較します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
int __far   strcmp(const char __near *s1, const char __near *s2);
int __far   _COM_strcmp_ff(const char __far *s1, const char __far *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 より小さい値を返します。

[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

文字数を指定文字数分比較します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
int __far   strncmp(const char __near *s1, const char __near *s2, size_t n);
int __far   _COM_strncmp_ff(const char __far *s1, const char __far *s2, size_t n);
```

[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 より小さい値を返します。

[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 *n* 文字比較します（null 文字に続く文字列は比較しない）。

memchr

オブジェクトから文字を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void __near * __far memchr(const void __near *s, int c, size_t n);
void __far * __far  _COM_memchr_f(const void __far *s, int c, size_t n);
```

[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 の指すオブジェクトの最初の *n* 個の文字の中に文字 *c* が最初に現れた位置を求めます。

strchr

文字列の先頭から文字を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strchr(const char __near *s, int c);
char __far * __far  _COM_strchr_f(const char __far *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]

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。

strcspn

文字列中の指定文字列を含まない先頭部分の長さを求めます。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
size_t __far    strcspn(const char __near *s1, const char __near *s2);
size_t __far    _COM_strcspn_ff(const char __far *s1, const char __far *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* の指す文字列の中にない文字のみで構成されている，先頭部分の最大の長さを求めます。

strpbrk

文字列から，指定したいずれかの文字を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strpbrk(const char __near *s1, const char __near *s2);
char __far * __far  _COM_strpbrk_ff(const char __far *s1, const char __far *s2);
```

[戻り値]

捜し出した文字へのポインタを返します。*s2* からの文字がいずれも *s1* の中に現れなかった場合は，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

文字列の最後から文字を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strrchr(const char __near *s, int c);
char __far * __far  _COM_strrchr_f(const char __far *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，null ポインタを返します。

[詳細説明]

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。

strspn

文字列中の指定文字列を含む先頭部分の長さを求めます。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
size_t __far    strspn(const char __near *s1, const char __near *s2);
size_t __far    _COM_strspn_ff(const char __far *s1, const char __far *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最大の長さを求めます。

strstr

文字列から文字列を探索します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __near * __far strstr(const char __near *s1, const char __near *s2);
char __far * __far  _COM_strstr_ff(const char __far *s1, const char __far *s2);
```

[戻り値]

見つかった文字列を指すポインタを返します。*s2* の指す文字列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。

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 __far * __far strtok(char __far *s1, const char __far *s2);（C90）
char __far * __far strtok(char __far * restrict s1, const char __far * restrict s2);（C99）【V1.07 以降】
```

[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]

s1 の指す文字列を，*s2* 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 *s1* 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。*s2* 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，*s2* 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 *s1* 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 *s1* 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。字句の次の探索はそこから開始されます。

第 1 実引数の値が null ポインタを持つ 2 回目以降の呼び出しでは，保持したポインタが指すところから探索が開始されます。この場合，リエントラントでないコードになります。

memset

オブジェクトの先頭から指定文字数分を指定文字で初期化します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
void __near * __far memset(const void __near *s, int c, size_t n);
void __far * __far  _COM_memset_f(const void __far *s, int c, size_t n);
```

[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 *n* 文字に *c* の値をコピーします。

[注意事項]

memset 関数は CC-RL が内部的に呼び出すことがあり，ランタイム・ライブラリに含まれます。同名のユーザ関数を作成しないでください。

strerror

エラー番号に対応したエラー・メッセージを取得します。

[所属]

標準ライブラリ

[指定形式]

```
#include <string.h>
char __far * __far  strerror(int errnum);
```

[戻り値]

errnum に対応するエラー・メッセージの文字列へのポインタを返します。
対応するエラー・メッセージが存在しない場合，null ポインタを返します。

[詳細説明]

エラー番号 *errnum* に対応するエラー・メッセージを取得します。エラー・メッセージは，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

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 __far    strlen(const char __near *s);
size_t __far    _COM_strlen_f(const char __far *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。

7.5.10 初期化処理関数

初期化処理関数として，以下のものがあります。

表 7.12 初期化処理関数

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
stkinit	スタック領域を初期化

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]

その他ライブラリ

[指定形式]

```
#include <_h_c_lib.h>
void __far      hdwinit(void);
```

[詳細説明]

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。

stkinit

スタック領域の初期化を行います。

[所属]

その他ライブラリ

[指定形式]

```
#include <_h_c_lib.h>
void __far      stkinit(void __near * stackbss);
```

[詳細説明]

スタック領域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。
引数には，スタック領域の先頭アドレス下位 16bit が渡されます。
未初期化 RAM 読み出しによるパリティ・エラー検出機能を使用しない場合は，本関数の呼び出しは不要です。

7.5.11 ランタイム・ライブラリ

ランタイム・ライブラリとして，以下のものがあります。

表 7.13 ランタイム・ライブラリ

関数名	概要
_COM_fadd	float 型加算
_COM_dadd	double 型（倍精度）加算
_COM_fsub	float 型減算
_COM_dsub	double 型（倍精度）減算
_COM_imul	int 型乗算
_COM_lmul	long 型乗算
_COM_llmul	long long 型乗算
_COM_fmul	float 型乗算
_COM_dmul	double 型（倍精度）乗算
_COM_mulsi	signed int 型乗算（結果は signed long 型）
_COM_mului	unsigned int 型乗算（結果は unsigned long 型）
_COM_mulsl	signed long 型乗算（結果は signed long long 型）
_COM_mulul	unsigned long 型乗算（結果は unsigned long long 型）
_COM_scdiv	signed char 型除算
_COM_ucdiv	unsigned char 型除算
_COM_sidiv	signed int 型除算
_COM_uidiv	unsigned int 型除算
_COM_sldiv	signed long 型除算
_COM_uldiv	unsigned long 型除算
_COM_slldiv	signed long long 型除算
_COM_ulldiv	unsigned long long 型除算
_COM_fdiv	float 型除算
_COM_ddiv	double 型（倍精度）除算
_COM_divui	unsigned int 型除算（除数は unsigned char 型）
_COM_divul	unsigned long 型除算（除数は unsigned int 型）
_COM_screm	signed char 型剰余算
_COM_ucrem	unsigned char 型剰余算
_COM_sirem	signed int 型剰余算
_COM_uirem	unsigned int 型剰余算
_COM_slrem	signed long 型剰余算
_COM_ulrem	unsigned long 型剰余算
_COM_sllrem	signed long long 型剰余算

関数名	概要
_COM_ullrem	unsigned long long 型剰余算
_COM_remui	unsigned int 型剰余算（除数は unsigned char 型）
_COM_remul	unsigned long 型剰余算（除数は unsigned int 型）
_COM_macsi	signed int 型積和演算（演算結果は signed long 型）
_COM_macui	unsigned int 型積和演算（演算結果は unsigned long 型）
_COM_lshl	long 型左シフト
_COM_llshl	long long 型左シフト
_COM_lshr	long 型論理右シフト
_COM_llshr	long long 型論理右シフト
_COM_lsar	long 型算術右シフト
_COM_llsar	long long 型算術右シフト
_COM_feq	float 型比較（==）
_COM_deq	double 型（倍精度）比較（==）
_COM_fne	float 型比較（!=）
_COM_dne	double 型（倍精度）比較（!=）
_COM_fge	float 型比較（>=）
_COM_dge	double 型（倍精度）比較（>=）
_COM_flt	float 型比較（<）
_COM_dlt	double 型（倍精度）比較（<）
_COM_fle	float 型比較（<=）
_COM_dle	double 型（倍精度）比較（<=）
_COM_fgt	float 型比較（>）
_COM_dgt	double 型（倍精度）比較（>）
_COM_funordered	float 型 NaN 判定
_COM_dunordered	double 型（倍精度）NaN 判定
_COM_sltof	signed long 型から float 型への型変換
_COM_sltod	signed long 型から double 型（倍精度）への型変換
_COM_ultof	unsigned long 型から float 型への型変換
_COM_ultod	unsigned long 型から double 型（倍精度）への型変換
_COM_slltof	signed long long 型から float 型への型変換
_COM_slltod	signed long long 型から double 型（倍精度）への型変換
_COM_ulltof	unsigned long long 型から float 型への型変換
_COM_ulltod	unsigned long long 型から double 型（倍精度）への型変換
_COM_ftosl	float 型から signed long 型への型変換
_COM_ftoul	float 型から unsigned long 型への型変換

関数名	概要
_COM_ftosll	float 型から signed long long 型への型変換
_COM_ftoull	float 型から unsigned long long 型への型変換
_COM_dtosl	double 型（倍精度）から signed long 型への型変換
_COM_dtoul	double 型（倍精度）から unsigned long 型への型変換
_COM_dtosll	double 型（倍精度）から signed long long 型への型変換
_COM_dtoull	double 型（倍精度）から unsigned long long 型への型変換
_COM_ftod	float 型から double 型（倍精度）への型変換
_COM_dtof	double 型（倍精度）から float 型への型変換
__control_flow_integrity【Professional 版のみ】【V1.06 以降】	間接関数呼び出しチェック

7.6 割り込み禁止時間，データ用セクションの使用，リエントラント性一覧

この節では，ライブラリに含まれている各種関数の割り込み禁止時間，初期値ありデータ用セクション（.data）の使用有無，初期値なしデータ用セクション（.bss）の使用有無，リエントラント性について説明します。

7.6.1 標準ライブラリ

以下に，標準ライブラリに含まれている各種関数の割り込み禁止時間，初期値ありデータ用セクション（.data）の使用有無，初期値なしデータ用セクション（.bss）の使用有無，リエントラント性を示します。
割り込み禁止時間は，左から乗除・積和演算器および乗除算拡張命令非使用時，乗除・積和演算器使用時，乗除算拡張命令使用時（単精度），乗除算拡張命令使用時（倍精度）を表します。数値の記述が 1 つしかないものは，各ライブラリで共通です。

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
assert	0	×	×	○	
isalnum	0	×	×	○	
isalpha	0	×	×	○	
isascii	0	×	×	○	
isblank 【V1.07 以降】	0	×	×	○	（C99）
iscntrl	0	×	×	○	
isdigit	0	×	×	○	
isgraph	0	×	×	○	
islower	0	×	×	○	
isprint	0	×	×	○	
ispunct	0	×	×	○	
isspace	0	×	×	○	
isupper	0	×	×	○	
isxdigit	0	×	×	○	
toascii	0	×	×	○	

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
tolower	0	×	×	○	
toupper	0	×	×	○	
imaxabs 【V1.07 以降】	0	×	×	○	（C99）
imaxdiv 【V1.07 以降】	0/42/0/0	×	×	○	（C99）
strtoimax 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
_COM_strtoimax_ff 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
strtoumax 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
_COM_strtoumax_ff 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
fpclassify 【V1.08 以降】	0	×	×	○	（C99）
isfinite 【V1.08 以降】	0	×	×	○	（C99）
isinf 【V1.08 以降】	0	×	×	○	（C99）
isnan 【V1.08 以降】	0	×	×	○	（C99）
isnormal 【V1.08 以降】	0	×	×	○	（C99）
signbit 【V1.08 以降】	0	×	×	○	（C99）
acos	0/41/0/0	○	×	×	errno
acosf	0/41/0/0	○	×	×	errno
acosl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
asin	0/41/0/0	○	×	×	errno
asinf	0/41/0/0	○	×	×	errno
asinl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
atan	0/41/0/0	○	×	×	errno
atanf	0/41/0/0	○	×	×	errno
atanl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
atan2	0/41/0/0	○	×	×	errno
atan2f	0/41/0/0	○	×	×	errno
atan2l 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
cos	0/14/0/0	○	×	×	errno
cosf	0/14/0/0	○	×	×	errno
cosl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
sin	0/14/0/0	○	×	×	errno
sinf	0/14/0/0	○	×	×	errno
sinl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
tan	0/41/0/0	○	×	×	errno

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
tanf	0/41/0/0	○	×	×	errno
tanl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
acosh 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
acoshf 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
acoshl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
asinh 【V1.08 以降】	0/41/0/0	○	×	○	（C99）
asinhf 【V1.08 以降】	0/41/0/0	○	×	○	（C99）
asinhl 【V1.08 以降】	0/41/0/0	○	×	○	（C99）
atanh 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
atanhf 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
atanhl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
cosh	0/41/0/0	○	×	×	errno
coshf	0/41/0/0	○	×	×	errno
coshl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
sinh	0/41/0/0	○	×	×	errno
sinhf	0/41/0/0	○	×	×	errno
sinhl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
tanh	0/41/0/0	○	×	×	errno
tanhf	0/41/0/0	○	×	×	errno
tanhl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
exp	0/41/0/0	○	×	×	errno
expf	0/41/0/0	○	×	×	errno
expl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
frexp	0/14/0/0	○	×	×	errno
frexpf	0/14/0/0	○	×	×	errno
frexpl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
ldexp	0/14/0/0	○	×	×	errno
ldexpf	0/14/0/0	○	×	×	errno
ldexpl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
log	0/14/0/0	○	×	×	errno
logf	0/14/0/0	○	×	×	errno
logl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
log10	0/14/0/0	○	×	×	errno
log10f	0/14/0/0	○	×	×	errno
log10l 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
log1p 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
log1pf 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
log1pl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
modf	0	○	×	×	errno
modff	0	○	×	×	errno
modfl 【V1.08 以降】	0	○	×	×	errno（C99）
scalbn 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
scalbnf 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
scalbnl 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
scalbln 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
scalblnf 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
scalblnl 【V1.09 以降】	0/14/0/0	○	×	×	errno（C99）
fabs	0	○	×	×	errno
fabsf	0	○	×	×	errno
fabsl 【V1.08 以降】	0	○	×	×	errno（C99）
pow	0/41/0/0	○	×	×	errno
powf	0/41/0/0	○	×	×	errno
powl 【V1.08 以降】	0/41/0/0	○	×	×	errno（C99）
sqrt	0/24/0/0	○	×	×	errno
sqrtf	0/24/0/0	○	×	×	errno
sqrtl 【V1.08 以降】	0/24/0/0	○	×	×	errno（C99）
ceil	0	○	×	×	errno
ceilf	0	○	×	×	errno
ceill 【V1.08 以降】	0	○	×	×	errno（C99）
floor	0	○	×	×	errno
floorf	0	○	×	×	errno
floorl 【V1.08 以降】	0	○	×	×	errno（C99）
nearbyint 【V1.09 以降】	0	×	×	○	（C99）
nearbyintf 【V1.09 以降】	0	×	×	○	（C99）
nearbyintl 【V1.09 以降】	0	×	×	○	（C99）
rint 【V1.09 以降】	0	×	×	○	（C99）
rintf 【V1.09 以降】	0	×	×	○	（C99）
rintl 【V1.09 以降】	0	×	×	○	（C99）
lrint 【V1.09 以降】	0	×	×	×	errno（C99）
lrintf 【V1.09 以降】	0	×	×	×	errno（C99）

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
lrintl 【V1.09 以降】	0	×	×	×	errno（C99）
llrint 【V1.09 以降】	0	×	×	×	errno（C99）
llrintf 【V1.09 以降】	0	×	×	×	errno（C99）
llrintl 【V1.09 以降】	0	×	×	×	errno（C99）
round 【V1.09 以降】	0	×	×	○	（C99）
roundf 【V1.09 以降】	0	×	×	○	（C99）
roundl 【V1.09 以降】	0	×	×	○	（C99）
lround 【V1.09 以降】	0	×	×	×	errno（C99）
lroundf 【V1.09 以降】	0	×	×	×	errno（C99）
lroundl 【V1.09 以降】	0	×	×	×	errno（C99）
llround 【V1.09 以降】	0	×	×	×	errno（C99）
llroundf 【V1.09 以降】	0	×	×	×	errno（C99）
llroundl 【V1.09 以降】	0	×	×	×	errno（C99）
trunc 【V1.09 以降】	0	×	×	○	（C99）
truncf 【V1.09 以降】	0	×	×	○	（C99）
truncl 【V1.09 以降】	0	×	×	○	（C99）
fmod	0/14/0/0	○	×	×	errno
fmodf	0/14/0/0	○	×	×	errno
fmodl 【V1.08 以降】	0/14/0/0	○	×	×	errno（C99）
copysign 【V1.09 以降】	0	×	×	○	（C99）
copysignf 【V1.09 以降】	0	×	×	○	（C99）
copysignl 【V1.09 以降】	0	×	×	○	（C99）
nan 【V1.09 以降】	0	×	×	○	（C99）
nanf 【V1.09 以降】	0	×	×	○	（C99）
nanl 【V1.09 以降】	0	×	×	○	（C99）
fdim 【V1.09 以降】	0	×	×	○	（C99）
fdimf 【V1.09 以降】	0	×	×	○	（C99）
fdiml 【V1.09 以降】	0	×	×	○	（C99）
fmax 【V1.09 以降】	0	×	×	○	（C99）
fmaxf 【V1.09 以降】	0	×	×	○	（C99）
fmaxl 【V1.09 以降】	0	×	×	○	（C99）
fmin 【V1.09 以降】	0	×	×	○	（C99）
fminf 【V1.09 以降】	0	×	×	○	（C99）
fminl 【V1.09 以降】	0	×	×	○	（C99）
isgreater 【V1.09 以降】	0	×	×	○	（C99）

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
isgreaterequal 【V1.09 以降】	0	×	×	○	（C99）
isless 【V1.09 以降】	0	×	×	○	（C99）
islessequal 【V1.09 以降】	0	×	×	○	（C99）
islessgreater 【V1.09 以降】	0	×	×	○	（C99）
isunordered 【V1.09 以降】	0	×	×	○	（C99）
setjmp	0	×	×	△	ポインタ参照先が更新された場合
longjmp	0	×	×	×	SP
va_start	0	×	×	○	
va_arg	0	×	×	○	
va_copy 【V1.09 以降】	0	×	×	○	（C99）
va_end	0	×	×	○	
printf	0/43/0/0	×	○	×	stdout，内部管理データ
scanf	0/41/0/0	○	×	×	stdin
snprintf 【V1.07 以降】	0/43/0/0	×	○	×	内部管理データ（C99）
sprintf	0/43/0/0	×	○	×	内部管理データ
sscanf	0/41/0/0	×	○	△	ポインタ参照先が更新された場合
vprintf	0/43/0/0	×	○	×	stdout，内部管理データ
vscanf 【V1.08 以降】	0/41/0/0	○	×	×	stdin（C99）
vsnprintf 【V1.07 以降】	0/43/0/0	×	○	×	内部管理データ（C99）
vsprintf	0/43/0/0	×	○	×	内部管理データ
vsscanf 【V1.08 以降】	0/41/0/0	○	×	△	ポインタ参照先が更新された場合（C99）
getchar	0	×	×	×	stdin
gets	0	×	×	×	stdin
_COM_gets_f	0	×	×	×	stdin
putchar	0	×	×	×	stdout
puts	0	×	×	×	stdout
_COM_puts_f	0	×	×	×	stdout
perror	0	○	×	×	errno
_COM_perror_f	0	○	×	×	errno
atof	0/41/0/0	○	×	×	errno
_COM_atof_f	0/41/0/0	○	×	×	errno
atoff	0/41/0/0	○	×	×	errno
_COM_atoff_f	0/41/0/0	○	×	×	errno
atoi	0/40/0/0	○	×	×	errno

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
_COM_atoi_f	0/40/0/0	○	×	×	errno
atol	0/40/0/0	○	×	×	errno
_COM_atol_f	0/40/0/0	○	×	×	errno
atoll 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
_COM_atoll_f 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
strtod	0/41/0/0	○	×	×	errno
_COM_strtod_ff	0/41/0/0	○	×	×	errno
strtof	0/41/0/0	○	×	×	errno
_COM_strtof_ff	0/41/0/0	○	×	×	errno
strtold 【V1.07 以降】	0/41/0/0	○	×	×	errno（C99）
_COM_strtold_ff 【V1.07 以降】	0/41/0/0	○	×	×	errno（C99）
strtol	0/40/0/0	○	×	×	errno
_COM_strtol_ff	0/40/0/0	○	×	×	errno
strtoll 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
_COM_strtoll_ff 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
strtoul	0/40/0/0	○	×	×	errno
_COM_strtoul_ff	0/40/0/0	○	×	×	errno
strtoull 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
_COM_strtoull_ff 【V1.07 以降】	0/43/0/0	○	×	×	errno（C99）
rand	0/24/0/0	○	×	×	seed
srand	0	○	×	×	seed
calloc 【V1.02 以降】	0	○	○	×	内部管理データ
free 【V1.02 以降】	0	○	○	×	内部管理データ
malloc 【V1.02 以降】	0	○	○	×	内部管理データ
realloc 【V1.02 以降】	0	○	○	×	内部管理データ
abort	0	×	×	-	処理が戻らないため
bsearch	0	×	×	△	ポインタ参照先が更新された場合
_COM_bsearch_f	0	×	×	△	ポインタ参照先が更新された場合
qsort	0/40/0/0	×	×	△	ポインタ参照先が更新された場合
_COM_qsort_f	0/40/0/0	×	×	△	ポインタ参照先が更新された場合
abs	0	×	×	○	
div	0/39/0/0	×	×	○	
labs	0	×	×	○	
ldiv	0/45/0/0	×	×	○	
llabs 【V1.07 以降】	0	×	×	○	（C99）

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
lldiv 【V1.07 以降】	0/42/0/0	×	×	○	（C99）
memcpy	0	×	×	△	ポインタ参照先が更新された場合
_COM_memcpy_f	0	×	×	△	ポインタ参照先が更新された場合
memmove	0	×	×	△	ポインタ参照先が更新された場合
_COM_memmove_ff	0	×	×	△	ポインタ参照先が更新された場合
strcpy	0	×	×	△	ポインタ参照先が更新された場合
_COM_strcpy_ff	0	×	×	△	ポインタ参照先が更新された場合
strncpy	0	×	×	△	ポインタ参照先が更新された場合
_COM_strncpy_ff	0	×	×	△	ポインタ参照先が更新された場合
strcat	0	×	×	△	ポインタ参照先が更新された場合
_COM_strcat_ff	0	×	×	△	ポインタ参照先が更新された場合
strncat	0	×	×	△	ポインタ参照先が更新された場合
_COM_strncat_ff	0	×	×	△	ポインタ参照先が更新された場合
memcmp	0	×	×	△	ポインタ参照先が更新された場合
_COM_memcmp_ff	0	×	×	△	ポインタ参照先が更新された場合
strcmp	0	×	×	△	ポインタ参照先が更新された場合
_COM_strcmp_ff	0	×	×	△	ポインタ参照先が更新された場合
strncmp	0	×	×	△	ポインタ参照先が更新された場合
_COM_strncmp_ff	0	×	×	△	ポインタ参照先が更新された場合
memchr	0	×	×	△	ポインタ参照先が更新された場合
_COM_memchr_f	0	×	×	△	ポインタ参照先が更新された場合
strchr	0	×	×	△	ポインタ参照先が更新された場合
_COM_strchr_f	0	×	×	△	ポインタ参照先が更新された場合
strcspn	0	×	×	△	ポインタ参照先が更新された場合
_COM_strcspn_ff	0	×	×	△	ポインタ参照先が更新された場合
strpbrk	0	×	×	△	ポインタ参照先が更新された場合
_COM_strpbrk_ff	0	×	×	△	ポインタ参照先が更新された場合
strrchr	0	×	×	△	ポインタ参照先が更新された場合
_COM_strrchr_f	0	×	×	△	ポインタ参照先が更新された場合
strspn	0	×	×	△	ポインタ参照先が更新された場合
_COM_strspn_ff	0	×	×	△	ポインタ参照先が更新された場合
strstr	0	×	×	△	ポインタ参照先が更新された場合
_COM_strstr_ff	0	×	×	△	ポインタ参照先が更新された場合
strtok	0	×	○	×	内部管理データ
memset	0	×	×	△	ポインタ参照先が更新された場合

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
_COM_memset_f	0	×	×	△	ポインタ参照先が更新された場合
strerror	0	×	×	○	
strlen	0	×	×	△	ポインタ参照先が更新された場合
_COM_strlen_f	0	×	×	△	ポインタ参照先が更新された場合
hdwinit	0	×	×	×	初期化処理
stkinit	0	×	×	×	初期化処理

7.6.2 ランタイム・ライブラリ

以下に，ランタイム・ライブラリに含まれている各種関数の割り込み禁止時間，初期値ありデータ用セクション（.data）の使用有無，初期値なしデータ用セクション（.bss）の使用有無，リエントラント性を示します。

割り込み禁止時間は，左から乗除・積和演算器および乗除算拡張命令非使用時，乗除・積和演算器使用時，乗除算拡張命令使用時（単精度），乗除算拡張命令使用時（倍精度）を表します。数値の記述が１つしかないものは，各ライブラリで共通です。

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
_COM_fadd	0	×	×	○	
_COM_dadd	0	×	×	○	
_COM_fsub	0	×	×	○	
_COM_dsub	0	×	×	○	
_COM_imul	0	×	×	○	
_COM_lmul	0/24/0/0	×	×	○	
_COM_llmul	0/24/0/0	×	×	○	
_COM_fmul	0/14/0/0	×	×	○	
_COM_dmul	0/14/0/0	×	×	○	
_COM_mulsi	0/14/0/0	×	×	○	
_COM_mului	0/14/0/0	×	×	○	
_COM_mulsl	0/14/0/0	×	×	○	
_COM_mulul	0/14/0/0	×	×	○	
_COM_scdiv	0/38/0/0	×	×	○	
_COM_ucdiv	0/38/0/0	×	×	○	
_COM_sidiv	0/38/0/0	×	×	○	
_COM_uidiv	0/38/0/0	×	×	○	
_COM_sldiv	0/40/0/0	×	×	○	
_COM_uldiv	0/40/0/0	×	×	○	
_COM_slldiv	0/43/0/0	×	×	○	
_COM_ulldiv	0/43/0/0	×	×	○	
_COM_fdiv	0/41/0/0	×	×	○	

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
_COM_ddiv	0	×	×	○	
_COM_divui	0/39/0/0	×	×	○	
_COM_divul	0/40/0/0	×	×	○	
_COM_screm	0/38/0/0	×	×	○	
_COM_ucrem	0/38/0/0	×	×	○	
_COM_sirem	0/38/0/0	×	×	○	
_COM_uirem	0/38/0/0	×	×	○	
_COM_slrem	0/40/0/0	×	×	○	
_COM_ulrem	0/40/0/0	×	×	○	
_COM_sllrem	0/42/0/0	×	×	○	
_COM_ullrem	0/42/0/0	×	×	○	
_COM_remui	0/39/0/0	×	×	○	
_COM_remul	0/39/0/0	×	×	○	
_COM_macsi	0/19/16/16	×	×	○	
_COM_macui	0/19/16/16	×	×	○	
_COM_lshl	0	×	×	○	
_COM_llshl	0	×	×	○	
_COM_lshr	0	×	×	○	
_COM_llshr	0	×	×	○	
_COM_lsar	0	×	×	○	
_COM_llsar	0	×	×	○	
_COM_feq	0	×	×	○	
_COM_deq	0	×	×	○	
_COM_fne	0	×	×	○	
_COM_dne	0	×	×	○	
_COM_fge	0	×	×	○	
_COM_dge	0	×	×	○	
_COM_flt	0	×	×	○	
_COM_dlt	0	×	×	○	
_COM_fle	0	×	×	○	
_COM_dle	0	×	×	○	
_COM_fgt	0	×	×	○	
_COM_dgt	0	×	×	○	
_COM_funordered	0	×	×	○	
_COM_dunordered	0	×	×	○	

関数名	割り込み禁止時間	.data 使用	.bss 使用	リエントラント性	備考（非リエントラント性の要因）
_COM_sltof	0	×	×	○	
_COM_sltod	0	×	×	○	
_COM_ultof	0	×	×	○	
_COM_ultod	0	×	×	○	
_COM_slltof	0	×	×	○	
_COM_slltod	0	×	×	○	
_COM_ulltof	0	×	×	○	
_COM_ulltod	0	×	×	○	
_COM_ftosl	0	×	×	○	
_COM_ftoul	0	×	×	○	
_COM_ftosll	0	×	×	○	
_COM_ftoull	0	×	×	○	
_COM_dtosl	0	×	×	○	
_COM_dtoul	0	×	×	○	
_COM_dtosll	0	×	×	○	
_COM_dtoull	0	×	×	○	
_COM_ftod	0	×	×	○	
_COM_dtof	0	×	×	○	
__control_flow_integrity【Professional 版のみ】【V1.06 以降】	0	×	×	○	

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。
作成したプログラムを実行させるには，必ずそのプログラムに応じたスタートアップ・ルーチンを作成しなければなりません。

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。

- リセット・ベクタテーブルの設定
- レジスタ・バンクの設定
- ミラー領域の設定
- スタック・ポインタの設定，スタック領域の初期化
- main 関数実行前に行う必要のある周辺 I/O レジスタの初期化
- RAM 領域セクションの初期化処理
- main 関数の起動
- 終了ルーチンの作成
- RL78-S1 コア用スタートアップ

システムによっては必要のない処理もあるので，それらに関しては省略できます。
なお，これらの処理は，基本的にアセンブラ命令で記述する必要があります。

8.2.1 リセット・ベクタテーブルの設定

リセット（リセット割り込み）が入った時の処理を記述します。RL78 では，リセットが入るとデバイスの設定によって決められたアドレス（リセット・アドレス）に格納されたアドレスに分岐します。そこで，リセット・アドレスにスタートアップ・ルーチンの先頭アドレスを設定します。記述は次のようになります。

```
_start  .VECTOR 0
```

また，スタートアップ・ルーチンの先頭は次のように記述します。

```
.SECTION        .text, TEXT
_start:
```

例ではスタートアップ・ルーチンの先頭のラベルを "_start" としていますが，別の名前でも問題はありません。

8.2.2 レジスタ・バンクの設定

リセット時にレジスタ・バンクは RB0 に初期化されるため，ワーク・レジスタを RB0 以外に設定したい場合のみ，以下のようにレジスタ・バンクの設定を行ってください。
SEL 命令がない RL78-S1 コアでは，本項の処理は記述できません。

```
SEL     RB1
```

8.2.3 ミラー領域の設定

RL78-S1 コアではプロセッサ・モード・コントロール・レジスタ（PMC）の値を変更できないため設定する必要はありません。RL78-S2 コア，RL78-S3 コアではリセット時に PMC の値は 0 になるため，PMC のビット MAA の値を変更したい場合のみ設定してください。
ミラー領域および PMC，MAA 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。

```
ONEB    !PMC
```

8.2.4 スタック・ポインタの設定，スタック領域の初期化

スタック領域の先頭 (__STACK_ADDR_END) および終端 (__STACK_ADDR_START) はリンカが決定します。CC-RL のスタック領域はアドレスの 0x0 番地方向に成長するので，スタック・ポインタにはシステムで使用するスタック領域の終端 (__STACK_ADDR_START) を設定します。__STACK_ADDR_START および __STACK_ADDR_END については「6.2.2 オプション指定により生成するシンボル」を参照してください。

スタートアップ・ルーチン内で SP をセットするには，次のように記述します。

```
MOVW    SP,#LOWW(__STACK_ADDR_START)
```

続けて，スタック領域を初期化します。引数として，スタック領域の先頭 (__STACK_ADDR_END) を渡します。未初期化 RAM 読み出しによるパリティ・エラー検出機能を使用しない場合，または同機能がないデバイスでは，スタック領域の初期化は必要ありません。スタック領域の初期化をコメントアウトしてください。未初期化 RAM 読み出しによるパリティ・エラー検出機能を使用する場合は，スタック領域の初期化処理を有効にしてください。

```
MOVW    AX,#LOWW(__STACK_ADDR_END)
CALL    !!_stkinit
```

8.2.5 main 関数実行前に行う必要のある周辺 I/O レジスタの初期化

スタートアップ・ルーチンを実行する上で，設定しなくてはならない周辺 I/O レジスタを初期化します。
なお，レジスタ設定は，アセンブラ命令でそのまま記述することも可能ですが，いったんスタートアップ・ルーチンから C 言語関数へ分岐し，その C 言語関数内で行う事も可能です。例えば，C 言語関数 “void hdwinit(void)” を作成して，スタートアップ・ルーチンから呼び出す時は，スタートアップ・ルーチン内に次の命令を記述します。

```
CALL    !!_hdwinit
```

8.2.6 RAM 領域セクションの初期化処理

初期値を持つ領域である “data 属性 ” 領域へ初期値のコピーと，初期値を持たない領域である “bss 属性 ” 領域のゼロ・クリアを行います。プログラム実行前に初期化するべき領域がない場合，本処理は必要ありません。

最初に，最適化リンカの -rom オプションで初期値データの生成を指示し，コピー先の RAM 領域セクションを定義します。詳細は「8.4 ROM イメージの作成」を参照してください。
定義した RAM 領域セクションを，スタートアップ・ルーチン内に次のように記述します。

```
.SECTION        .dataR, DATA
.SECTION        .sdataR, DATA
```

次に，スタートアップ・ルーチン内で，bss 属性の領域を初期化するコードを記述します。
セクション .bss，.sbss を 0 初期化する場合，次のように記述します。

```
        ; clear external variables which doesn't have initial value (near)
        MOVW    HL,#LOWW(STARTOF(.bss))
        MOVW    AX,#LOWW(STARTOF(.bss) + SIZEOF(.bss))
        BR      $.L2_BSS
.L1_BSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_BSS:
        CMPW    AX,HL
        BNZ     $.L1_BSS

        ; clear saddr variables which doesn't have initial value
        MOVW    HL,#LOWW(STARTOF(.sbss))
        MOVW    AX,#LOWW(STARTOF(.sbss) + SIZEOF(.sbss))
        BR      $.L2_SBSS
.L1_SBSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_SBSS:
        CMPW    AX,HL
        BNZ     $.L1_SBSS
```

次に，スタートアップ・ルーチン内で，data 属性の領域を RAM 領域セクションにコピーするコードを記述します。
セクション .data，.sdata をそれぞれ .dataR，.sdataR にコピーする場合，次のように記述します。このコピールーチンは，64Kbyte 境界をまたぐセクションについては対応していないので注意してください。

```
        ; copy external variables having initial value (near)
        MOV     ES,#HIGHW(STARTOF(.data))
        MOVW    BC,#LOWW(SIZEOF(.data))
        BR      $.L2_DATA
.L1_DATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.data))[BC]
        MOV     LOWW(STARTOF(.dataR))[BC],A
.L2_DATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_DATA

        ; copy saddr variables having initial value
        MOV     ES,#HIGHW(STARTOF(.sdata))
        MOVW    BC,#LOWW(SIZEOF(.sdata))
        BR      $.L2_SDATA
.L1_SDATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.sdata))[BC]
        MOV     LOWW(STARTOF(.sdataR))[BC],A
.L2_SDATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_SDATA
```

RAM 領域セクションへの 0 初期化 , およびコピー処理は ,C 言語で作成しスタートアップ・ルーチンから呼ぶこともできます。

```
        ; initializing RAM section
        CALL    !!_INITSCT_RL
```

C 言語での RAM 領域セクション初期化関数の例を以下に示します。

```
#define BSEC_MAX        2               /*0 初期化する BSS セクションの数 */
#define DSEC_MAX        2               /* コピーする DATA セクションの数 */

const struct bsec_t {
        char __near *ram_sectop;        /* セクション先頭アドレス */
        char __near *ram_secend;        /* セクション末尾アドレス +1*/
} bsec_table[BSEC_MAX] = {
        {(char __near *)__sectop(".bss"),
         (char __near *)__secend(".bss")},
        {(char __near *)__sectop(".sbss"),
         (char __near *)__secend(".sbss")}};

const struct dsec_t {
        char __far *rom_sectop;         /* コピー元セクション先頭アドレス */
        char __far *rom_secend;         /* コピー元セクション末尾アドレス +1*/
        char __near *ram_sectop;        /* コピー先セクション先頭アドレス */
} dsec_table[DSEC_MAX] = {
        {__sectop(".data"),
         __secend(".data"),
         (char __near *)__sectop(".dataR")},
        {__sectop(".sdata"),
         __secend(".sdata"),
         (char __near *)__sectop(".sdataR")}};

void INITSCT_RL(void)
{
        unsigned int i;
        char __far *rom_p;
        char __near *ram_p;

        for (i = 0; i < BSEC_MAX; i++) {
                ram_p = bsec_table[i].ram_sectop;
                for ( ; ram_p != bsec_table[i].ram_secend; ram_p++) {
                        *ram_p = 0;
                }
        }
        for (i = 0; i < DSEC_MAX; i++) {
                rom_p = dsec_table[i].rom_sectop;
                ram_p = dsec_table[i].ram_sectop;
                for ( ; rom_p != dsec_table[i].rom_secend; rom_p++, ram_p++) {
                        *ram_p = *rom_p;
                }
        }
}
```

8.2.6.1 初期化テーブルを利用した RAM 領域セクションの初期化処理【V1.12 以降】

V1.12 以降では，リンカに -ram_init_table_section を指定することにより，RAM 領域セクションを初期化するための情報を持つテーブルを実行形式に埋め込むことができます。
テーブルの各レコード（行）は初期化対象のセクションに対応し，次のフィールドを持ちます。

	サイズ	初期値を持つセクション	初期値を持たないセクション	終端レコード
フィールド 1	4byte	初期値データの先頭アドレス	セクション先頭アドレス	0
フィールド 2	2byte	セクション・サイズ	セクション・サイズ	0
フィールド 3	2byte	セクション先頭アドレス	セクション先頭アドレス	0

詳細は，リンク・オプションの -ram_init_table_section を参照してください。

```
> rlink a.obj b.obj -form=absolute -output=a.abs -rom=.data=.dataR
  -rom=.sdata=.sdataR -ram_init_table_section
```

リンカが生成したテーブルを用いて RAM を初期化するには，前述の bss 属性領域の初期化コード，data 属性領域の初期化コードのかわりに，次のようなコードを使用します。

```
  MOVW DE, #LOWW(STARTOF(.ram_init_table))
  MOV A, #LOW(HIGHW(STARTOF(.ram_init_table)))
  PUSH AX
  PUSH DE

.TABLE_LOOP:
  MOV A, [SP+0x03]
  MOV ES, A
  MOVW AX, [SP+0x00]
  MOVW DE, AX

  CLRB B

  # src_lo
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # src_hi
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # size
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  # dest
  CALL $!.GET_RAM_INIT_RECORD
  PUSH AX

  CMP0 B
  BZ $.RETURN

  MOVW AX, DE
  MOVW [SP+0x08], AX
  MOV A, ES
  MOV [SP+0x0B], A
```

```
  # dest
  POP DE
  # size
  POP BC
  # src_hi
  POP AX
  MOV A, X
  MOV ES, A
  # src_lo
  POP HL

  MOVW AX, DE
  CMPW AX, HL
  BNZ $.COPY
  MOV A, ES
  CMP A, #0xF
  BNZ $.COPY

.CLEAR:
  MOVW AX, BC
  OR A, X
  BZ $.TABLE_LOOP
  DECW BC
  CLRB A
  MOV [DE], A
  INCW DE
  BR $.CLEAR

.COPY:
  MOVW AX, BC
  OR A, X
  BZ $.TABLE_LOOP
  DECW BC
  MOV A, ES:[HL]
  MOV [DE], A
  INCW HL
  INCW DE
  BR $.COPY

.RETURN:
  ADDW SP, #0x0C
  RET

.GET_RAM_INIT_RECORD:
  MOVW AX, ES:[DE]
  MOVW HL, AX

  OR A, X
  OR B, A

  MOVW AX, DE
  ADDW AX, #0x0002
  MOVW DE, AX
  MOV A, ES
  ADDC A, #0
  MOV ES, A

  MOVW AX, HL
  RET
```

C 言語で初期化関数を作成して，スタートアップ・ルーチン内から呼び出すことも可能です。
ただし，スタック領域を初期化しないよう注意してください。
スタートアップ・ルーチン内：

```
CALL !!_ram_init
```

初期化関数：

```
typedef unsigned char __far * src_ptr;
typedef unsigned short src_len;
typedef unsigned char __near * dest_ptr;
 struct table {
 src_ptr src;
 src_len len;
 dest_ptr dest;
};
typedef struct table __far * table_ptr;

void ram_init(void)
{
 table_ptr record;
 for(record = __sectop(".ram_init_table"); ; record++)
 {
  src_ptr src = record->src;
  src_len len = record->len;
  dest_ptr dest = record->dest;

  if(src == 0 && len == 0 && dest == 0) /* END OF RECORD */
   break;

  if(src == dest)
  {
   /* RAM CLEAR */
   while(len--)
   {
    *dest = 0;
    dest++;
   }
  }
  else
  {
   /* ROM -> RAM COPY */
   while(len--)
   {
    *dest = *src;
    src++;
    dest++;
   }
  }
 }
}
```

8.2.7 main 関数の起動

スタートアップ・ルーチンで行う必要のある処理がすべて終わった時に，main 関数を呼び出します。
main 関数を呼び出す時は，次の命令を記述します。

```
CALL    !!_main
```

8.2.8 終了ルーチンの作成

何もせず無限ループさせるには，次のように記述してください。

```
_exit:
        BR      $_exit
```

8.2.9 RL78-S1 コア用スタートアップ

ROM/RAM が小さい RL78-S1 コア用のスタートアップは，コード効率化を考慮し，スタック領域や bss 属性の領域を初期化する代わりに RAM 全体を初期化しています。RAM の先頭アドレス (__RAM_ADDR_START) と終了アドレス（RAM_ADDR_END）はリンカが決定します。__RAM_ADDR_START，および __RAM_ADDR_END については「6.2.2 オプション指定により生成するシンボル」を参照してください。

```
        ;-------------------------------------------------
        ; initializing RAM
        ;-------------------------------------------------
        MOVW    HL,#LOWW(__RAM_ADDR_START)
        MOVW    AX,#LOWW(__RAM_ADDR_END)
        BR      $.L2_RAM
.L1_RAM:
        MOV     [HL+0],#0
        INCW    HL
.L2_RAM:
        CMPW    AX,HL
        BNZ     $.L1_RAM
```

8.3 コーディング例

スタートアップ・ルーチンの例を，次に示します。

表 8.1　スタートアップ・ルーチンの例

```
; Copyright (C) 2014 Renesas Electronics Corporation
; RENESAS ELECTRONICS CONFIDENTIAL AND PROPRIETARY.
; This program must be used solely for the purpose for which
; it was furnished by Renesas Electronics Corporation. No part of this
; program may be reproduced or disclosed to others, in any
; form, without the prior written permission of Renesas Electronics
; Corporation.

; NOTE : THIS IS A TYPICAL EXAMPLE

        .public _start
        .public _exit
;-------------------------------------------------------------------------------
; RAM section
;-------------------------------------------------------------------------------
.SECTION        .dataR, DATA
.SECTION        .sdataR, DATA
```

```
;-----------------------------------------------------------------------
; RESET vector
;-----------------------------------------------------------------------
_start  .VECTOR 0
;-----------------------------------------------------------------------
; startup
;-----------------------------------------------------------------------
.SECTION        .text, TEXT
_start:
        ;------------------------------------------------
        ; setting the stack pointer
        ;------------------------------------------------
        MOVW    SP,#LOWW(__STACK_ADDR_START)

        ;------------------------------------------------
        ; initializing stack area
        ;------------------------------------------------
        MOVW    AX,#LOWW(__STACK_ADDR_END)
        CALL    !!_stkinit

        ;------------------------------------------------
        ; hardware initialization
        ;------------------------------------------------
        CALL    !!_hdwinit
$IFNDEF __USE_RAM_INIT_TABLE

        ;------------------------------------------------
        ; initializing BSS
        ;------------------------------------------------
        ; clear external variables which doesn't have initial value (near)
        MOVW    HL,#LOWW(STARTOF(.bss))
        MOVW    AX,#LOWW(STARTOF(.bss) + SIZEOF(.bss))
        BR      $.L2_BSS
.L1_BSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_BSS:
        CMPW    AX,HL
        BNZ     $.L1_BSS

        ; clear saddr variables which doesn't have initial value
        MOVW    HL,#LOWW(STARTOF(.sbss))
        MOVW    AX,#LOWW(STARTOF(.sbss) + SIZEOF(.sbss))
        BR      $.L2_SBSS
.L1_SBSS:
        MOV     [HL+0],#0
        INCW    HL
.L2_SBSS:
        CMPW    AX,HL
        BNZ     $.L1_SBSS

        ;------------------------------------------------
        ; ROM data copy
        ;------------------------------------------------
        ; copy external variables having initial value (near)
        MOV     ES,#HIGHW(STARTOF(.data))
        MOVW    BC,#LOWW(SIZEOF(.data))
        BR      $.L2_DATA
```

```
.L1_DATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.data))[BC]
        MOV     LOWW(STARTOF(.dataR))[BC],A
.L2_DATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_DATA
        ; copy saddr variables having initial value
        MOV     ES,#HIGHW(STARTOF(.sdata))
        MOVW    BC,#LOWW(SIZEOF(.sdata))
        BR      $.L2_SDATA
.L1_SDATA:
        DECW    BC
        MOV     A,ES:LOWW(STARTOF(.sdata))[BC]
        MOV     LOWW(STARTOF(.sdataR))[BC],A
.L2_SDATA:
        CLRW    AX
        CMPW    AX,BC
        BNZ     $.L1_SDATA

$ELSE
        MOVW DE, #LOWW(STARTOF(.ram_init_table))
        MOV A, #LOW(HIGHW(STARTOF(.ram_init_table)))

        CALL $!_ram_init

$ENDIF

        ;-------------------------------------------------
        ; call main function
        ;-------------------------------------------------
        CALL    !!_main          ; main();

        ;-------------------------------------------------
        ; call exit function
        ;-------------------------------------------------
        CLRW    AX               ; exit(0);
_exit:
        BR      $_exit

$IFDEF __USE_RAM_INIT_TABLE
# A,DE: FAR ADDRESS
_ram_init:
        PUSH AX
        PUSH DE

.TABLE_LOOP:
        MOV A, [SP+0x03]
        MOV ES, A
        MOVW AX, [SP+0x00]
        MOVW DE, AX

        CLRB B

        # src_lo
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX
```

```
        # src_hi
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        # size
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        # dest
        CALL $!.GET_RAM_INIT_RECORD
        PUSH AX

        CMP0 B
        BZ $.RETURN

        MOVW AX, DE
        MOVW [SP+0x08], AX
        MOV A, ES
        MOV [SP+0x0B], A

        # dest
        POP DE
        # size
        POP BC
        # src_hi
        POP AX
        MOV A, X
        MOV ES, A
        # src_lo
        POP HL

        MOVW AX, DE
        CMPW AX, HL
        BNZ $.COPY
        MOV A, ES
        CMP A, #0xF
        BNZ $.COPY

.CLEAR:
        MOVW AX, BC
        OR A, X
        BZ $.TABLE_LOOP
        DECW BC
        CLRB A
        MOV [DE], A
        INCW DE
        BR $.CLEAR

.COPY:
        MOVW AX, BC
        OR A, X
        BZ $.TABLE_LOOP
        DECW BC
        MOV A, ES:[HL]
        MOV [DE], A
        INCW HL
        INCW DE
        BR $.COPY
```

```
.RETURN:
        ADDW SP, #0x0C
        RET

.GET_RAM_INIT_RECORD:
        MOVW AX, ES:[DE]
        MOVW HL, AX

        OR A, X
        OR B, A

        MOVW AX, DE
        ADDW AX, #0x0002
        MOVW DE, AX
        MOV A, ES
        ADDC A, #0
        MOV ES, A

        MOVW AX, HL
        RET

$ENDIF

;-------------------------------------------------------------------------------
; section
;-------------------------------------------------------------------------------
.SECTION .RLIB, TEXTF
.L_section_RLIB:
.SECTION .SLIB, TEXTF
.L_section_SLIB:
.SECTION .textf, TEXTF
.L_section_textf:
.SECTION .const, CONST            ; ミラー空間ありデバイスの場合のみ
.L_section_const:                 ; ミラー空間ありデバイスの場合のみ
.SECTION .constf, CONSTF
.L_section_constf:
.SECTION .data, DATA
.L_section_data:
.SECTION .sdata, SDATA
.L_section_sdata:
.SECTION .bss, BSS
.L_section_bss:
.SECTION .sbss, SBSS
.L_section_sbss:
```

ミラー空間がないデバイス用のスタートアップは，const 属性の記述はありません。【V1.02 以降】

8.4 ROM イメージの作成

この節では，組み込み向けアプリケーションで必要となる，ROM イメージの作成について説明します。

アプリケーション中で外部変数や静的変数を定義すると，それらの変数は RAM 上のセクションに配置されます。変数が初期値を持つ場合，アプリケーションの開始時には RAM に初期値が存在している必要があります。
一方で，ハードウェアの起動時やリセット時は RAM の値は不定です。従って，リセットからアプリケーションの起動までの間に，RAM に変数の初期値を格納する必要があります。
CC-RL では，プログラム本体と初期値データを ROM に配置した，ROM だけのプログラムイメージを作成することができます。このプログラムを実行すると，スタートアップ・ルーチン内で，ROM に準備した初期値データを RAM にコピーして RAM の初期化を行います。
セルフ・プログラミングのために，プログラム・コードを RAM 上に配置することがあります。この場合も同様に，スタートアップ・ルーチン内で，ROM に準備したプログラム・コードを RAM に配置します。

ROM から RAM へデータを配置するには，最適化リンカの -rom オプションを使用します。

```
-rom=初期値データセクション名=初期化先セクション名
```

リンカでセクションの自動配置機能を指定しない場合は，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。リンカでセクションの自動配置機能を指定する場合は，-start オプションでのアドレス指定は不要です。

```
-start=セクション名[,セクション名]/配置先アドレス
```

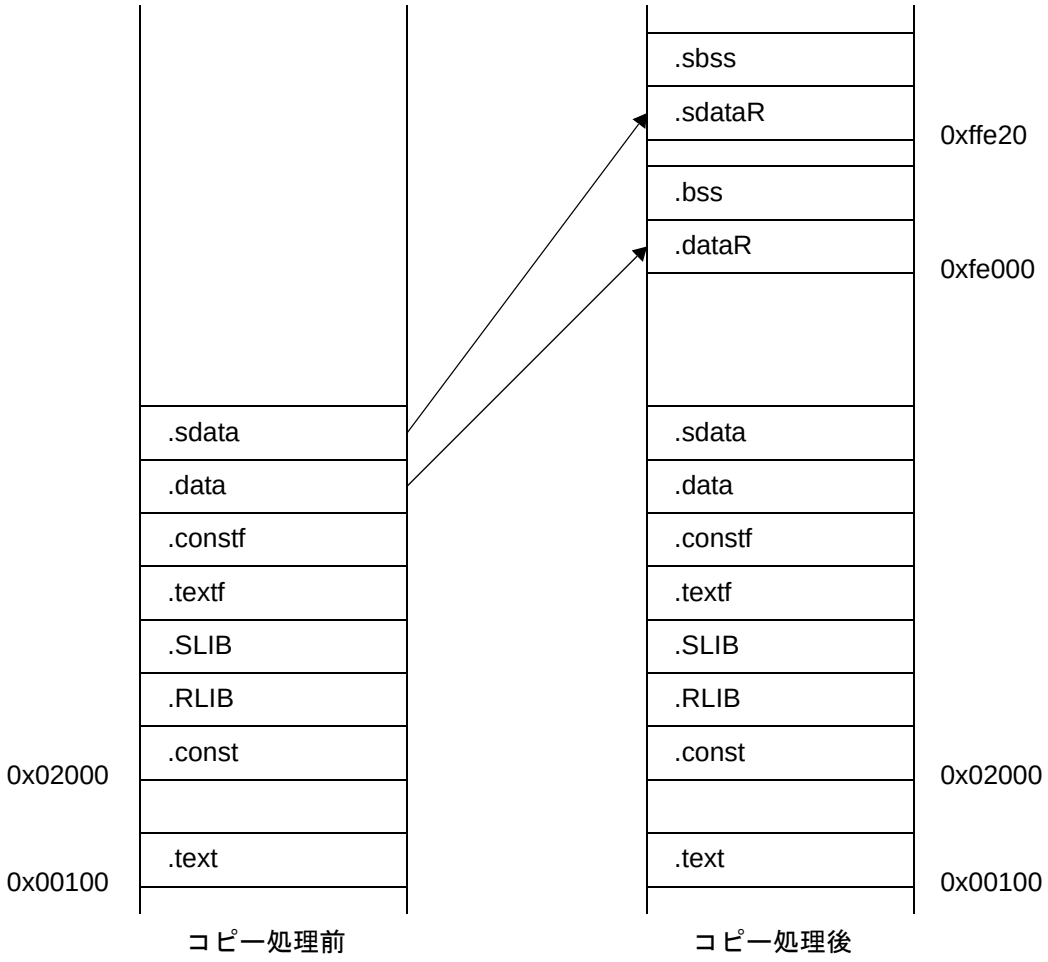
ユーザ・プログラム中にセクション .text，.textf，.RLIB，.SLIB，.const，.constf，.data，.sdata，.bss，.sbss があるとします。実行時に .text を 0x100 番地に，.const を 0x2000 番地に，再配置属性 DATA,BSS のセクションを 0xfe000 番地に，再配置属性 SDATA,SBSS のセクションを 0xffe20 番地に配置したい場合，次のように記述します。

```
-start=.text/100
-start=.const,.RLIB,.SLIB,.textf,.constf,.data,.sdata/2000
-start=.dataR,.bss/FE000
-start=.sdataR,.sbss/FFE20
-rom=.data=.dataR,.sdata=.sdataR
```

コピー先のセクション名 .dataR，.sdataR は別名でも構いませんが，スタートアップ内の記述と合わせる必要があります。実行時には，.data から .dataR，.sdata から .sdataR にデータがコピーされ，.bss，.sbss を初期化して使用します。

配置イメージは次のようになります。

図 8.1 コピー処理前後の配置イメージ



9. 関数呼び出し仕様

この章では，CC-RL 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，C ソース・プログラムとアセンブリ・ソース・プログラムの間で相互に関数呼び出しを行う際に，アセンブリ・プログラム側で留意すべき点について説明します。

9.1.1 値が保証される汎用レジスタおよび ES/CS レジスタ

ここでは，関数呼び出し前後や割り込み前後で，レジスタの値が同一であることを保証するレジスタ，および保証しないレジスタについて説明します。

表 9.1 各種レジスタの値と各種条件の関係

レジスタの種類	関数呼び出しの前後でレジスタの値が同一であることを保証するか	割り込み発生の前後でレジスタの値が同一であることを保証するか
汎用レジスタ AX，BC，DE，HL	しない	する
ES，CS レジスタ	しない	する
PSW，PC レジスタ	しない	する
MACR レジスタ	する	する
その他のレジスタ	しない	しない

9.1.1.1 汎用レジスタ AX，BC，DE，HL

本仕様は，汎用レジスタ AX，BC，DE，HL の各々について，関数呼び出しの前後で値が同一であることを保証しません。
割り込み発生の前後の場合は，利用中のレジスタ・バンクに属する汎用レジスタに限り，同一であることを保証します。保証するための手段には次の 2 種類があります。

- スタック領域を使用する方法
  割り込み発生時にスタック領域にレジスタの値を退避し，割り込み終了時にスタック領域から値を復帰します。後述する，汎用レジスタ以外のレジスタの場合もこの方法を使用します。
- レジスタ・バンク切り替え機能を使用する方法
  割り込み用プラグマ指令のレジスタ・バンク指定機能を使うと，スタック領域を使わずに，レジスタ・バンクの切り替えにより効率よく汎用レジスタの値の退避，復帰を行うことができます。
  レジスタ・バンクの切り替えを使用する場合の注意事項を次に示します。これらの注意事項に該当する場合，割り込み発生前後でレジスタの値が同一であることは保証しません。
  - 割り込み用プラグマ指令を使わず，割り込みハンドラ内でレジスタ・バンクを切り替えてはいけない。
  - 割り込みハンドラ内から，return 文以外の方法で復帰してはいけない。
  - 割り込み発生元と同じレジスタ・バンクへ切り替えてはいけない。
    多重割り込みである場合，最初の割り込み発生元，および発生元へ復帰するまでのすべての割り込みハンドラが，異なるレジスタ・バンクを使用する必要がある。

9.1.1.2 ES，CS レジスタ

ES，CS レジスタの各々について，関数呼び出しの前後で値が同一であることを保証しません。
割り込み発生の前後の場合は同一であることを保証します。

9.1.1.3 PSW，PC レジスタ

PSW，PC レジスタの各々について，関数呼び出しの前後で値が同一であることを保証しません。
割り込み発生の前後の場合は同一であることを保証します。なお，値の退避は割り込み発生時にハードウェアによって自動的に行われ，復帰は割り込み終了命令により行われます。従って，割り込みハンドラ内で明示的に退避，復帰する必要はありません。

9.1.1.4 MACR レジスタ

MACR レジスタについては，関数呼び出しの前後で，レジスタの値が同一であることを保証します。
割り込み発生の前後においても，レジスタの値が同一であることを保証します。
ただし，コンパイル・オプションの -use_mach=not_use を指定した場合（V1.10 以前では常にこの状態です）は，コンパイラの生成コードは MACR レジスタの使用[注]も管理も行いません。MACR レジスタを使用する場合は，プログラム内で自主的に管理する必要があります。

注 このとき，一部の組み込み関数は，MACR レジスタを使用し，その値を書き替えます。

9.1.1.5 その他のレジスタ

上記に該当しないレジスタについては，関数呼び出しの前後においても，割り込み発生の前後においても，レジスタの値が同一であることを保証しません。

9.1.2 実引数の受け渡し

(1) 実引数の受け渡しに利用可能なレジスタ
実引数の受け渡しに利用可能なレジスタは AX，BC，および DE です。

(2) 実引数の割り付け対象と実引数の型・サイズの変更
実引数の割り付け対象と実引数の型，およびサイズをどのように変更するかを以下に示します。
ここに記述したもの以外は変更を受けません。

実引数の種類	実引数の割り付け対象	実引数の型・サイズの変更
関数原型が参照でき，仮引数の型が参照できる場合	レジスタ，またはスタック	far ポインタをレジスタに割り付ける場合は 3 バイト分のレジスタに割り付けます。
関数原型が参照でき，仮引数の型が参照できない場合（＝可変個引数）	スタック	既定の実引数拡張に従います。
関数原型が参照できない場合	レジスタ，またはスタック	既定の実引数拡張に従います。 既定の実引数拡張の結果，far ポインタをレジスタに割り付ける場合は 3 バイト分のレジスタに割り付けます。

注意 1. 変数のサイズや既定の実引数拡張については「(7) 既定の実引数拡張」を参照してください。

注意 2. 可変個引数の直前の実引数は「関数原型が参照でき，仮引数の型が参照できる場合」として扱います。

(3) 実引数のレジスタへの割り付け
実引数のレジスタへの割り付けは，以下のようになります。

- レジスタに割り付ける候補となるのは 4 バイト以下のサイズの実引数です。
ただ，構造体型，および共用体型の実引数がレジスタに割り付く場合，構造体型，および共用体型に含まれるパディングもレジスタに割り付けられます。
- 実引数が構造体，または共用体の場合，すべてのメンバがレジスタに割り付けられるか，スタックに割り付けられるかのどちらか一方です。
- far ポインタはその下位 3 バイトをレジスタに割り付けます。
far ポインタにおけるページ番号部分（＝ far アドレス 20 ビットの上位 4 ビット）は下位 3 バイトのうちの上位 1 バイトが割り付くレジスタの下位 4 ビットに格納します。
例えば，far ポインタを A-DE に割り付ける場合，ページ番号部分は A レジスタの下位 4 ビットに格納されます。

- 実引数は第 1 引数から最後の引数の順（プログラム上，左から右の順）に，利用可能なレジスタのうち最も優先順位の高いレジスタに割り付けられます。利用可能なレジスタがない場合，その実引数はスタックに割り付けられます。優先順位を以下に示します。

実引数のサイズ	割り付けるレジスタの優先順位（左側ほど優先順位が高い）
1 バイト	A，X，C，B，E，D
2 バイト	AX，BC，DE
3 バイト	C-AX，X-BC，E-BC，X-DE，B-DE
4 バイト	far ポインタ： A-DE，X-DE，C-DE，B-DE，X-BC far ポインタ以外： BC-AX，DE-BC

この表で “-” は，8 ビット，または 16 ビット・レジスタとほかの 16 ビット・レジスタを結び付ける記号です。各引数のレジスタへの割り付けは，引数を構成するバイトのアドレス降順（大きいアドレスから小さいアドレスの順）が上記の表に指定されたレジスタの指定順（左から右の順）に一致するように割り付けられます。

例 1. void foo(char p1, short p2, char p3) を呼び出すとき，p1 を A に , p2 を BC に , p3 を X に割り付けます。

例 2. 以下の構造体型の実引数 S がレジスタに割り付く時，c1 は X に，s2 は BC に割り付けられます。A にはパディングが割り付きます。

```
struct {
        char    c1;
        short   s2;
} S;
```

例 3. void foo(long x) を呼び出すとき，x の上位 2 バイトは BC レジスタ，下位 2 バイトは AX レジスタに割り付けます。

例 4. 以下の構造体型の実引数 S3 がレジスタに割り付く時，最上位バイトは C レジスタ，次の 1 バイトは A レジスタ，最下位バイトは X レジスタに割り付けられます。

```
struct{
        char    a[3];
} S3;
```

注意 1. long long，および -dbl_size=8 の時の double 等の 8 バイト・データはスタックに割り付けられます。

注意 2. 5 バイト以上の構造体や共用体はスタックに割り付けられますが，4 バイト以下の構造体や共用体はレジスタに割り付ける候補となります。

(4) 実引数のスタックへの割り付け
実引数のスタックへの割り付けは，以下のようになります。

- スタックで渡す実引数自体はリトル・エンディアンで配置し，2 バイト境界にアラインされます。
- スタックで渡す実引数の配置順序は，実引数並びにおいて，より左側にある実引数がより小さいアドレスになるように配置されます。
- スタックで渡す実引数の配置は，実引数間に 1 バイトのパディングが入ることを除けばスタック内に連続して配置されます。
- 各引数のスタックへの割り付けは，引数を構成するバイトのアドレス降順（大きいアドレスから小さいアドレスの順）がスタックのアドレス降順（大きいアドレスから小さいアドレスの順）に一致するように割り付けられます。
- far ポインタは，その下位 3 バイトをスタック上の 4 バイトの領域に割り付けられます。4 バイトのうちの上位 1 バイトの値は不定です。far ポインタにおけるページ番号部分（＝ far アドレス 20 ビットの上位 4 ビット）は 4 バイトの領域中の上位から 2 バイト目の下位 4 ビットに割り付けられます。

注意 スタックへの配置の仕方は，「9.1.4 スタック・フレーム」を参照してください。

- スタックに割り付ける実引数は以下です。
  - 1 バイトから 4 バイトのサイズの実引数でレジスタに割り付けないもの
  - 5 バイト以上のサイズの実引数
  - 可変個の実引数

例 void foo(long long x) を呼び出すとき，スタック・ポインタを sp とすると，x の最上位 1 バイトは sp+7 が指す位置へ割り付け，以降，各バイトをアドレスの降順に sp+6，sp+5，sp+4，sp+3，sp+2，sp+1 へ割り付け，最下位バイトは sp+0 が指す位置へ割り付けられます。

9.1.3 戻り値

(1) 戻り値の受け渡しに利用可能なレジスタ
戻り値の受け渡しに利用可能なレジスタは AX，BC，および DE です。

(2) 戻り値，または戻り値へのポインタのレジスタ
戻り値，または戻り値へのポインタのレジスタは，以下のようになります。

- サイズが 5 バイト以上の戻り値の場合，戻り値へのポインタを near ポインタとして第 1 引数に設定されます。
  すなわち，戻り値には第 1 引数としての受け渡し規則が適用されます。これに伴い，プログラム上で指定された第 n 引数（n=1,…,N）には第 n+1 引数としての受け渡し規則が適用されます。
  詳細は「9.1.2 実引数の受け渡し」を参照してください。
- サイズが 4 バイト以下の戻り値はレジスタに割り付けられます。
- far ポインタのレジスタ割り付け方法に関しては，「9.1.2 実引数の受け渡し」を参照してください。
- メンバが far ポインタのみから構成されるサイズが 4 バイトの構造体，または共用体は，far ポインタ以外と見なされます。
- サイズが 4 バイト以下の戻り値のレジスタ割り付け規則を以下の表に示します。

戻り値のサイズ	割り付けるレジスタ
1 バイト	A
2 バイト	AX
3 バイト	C-AX
4 バイト	far ポインタ： A-DE far ポインタ以外： BC-AX

この表で “-” は，8 ビット，または 16 ビット・レジスタとほかの 16 ビット・レジスタを結び付ける記号です。
各戻り値のレジスタへの割り付けは，戻り値を構成するバイトのアドレス降順（大きいアドレスから小さいアドレスの順）が上記の表に指定されたレジスタの指定順（左から右の順）に一致するように割り付けられます。

注意 4 バイト以下の構造体や共用体はレジスタに割り付ける候補となります。

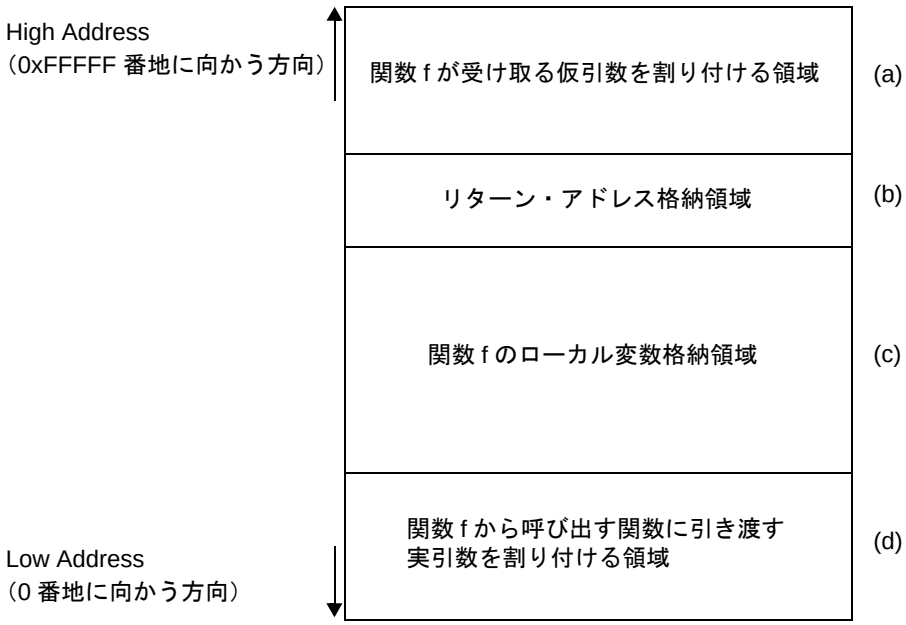
9.1.4 スタック・フレーム

(1) スタック・ポインタに設定する値
スタック・ポインタ（SP）には 2 の倍数のアドレスが設定されます。

(2) スタック・フレームの割り当てと解放
スタック・ポインタは常にスタック・フレームの最下位アドレスを指します。したがって，SP のアドレスよりも小さい領域に格納された値は保証されません。
関数の呼び出し時に呼び出し元関数で割り当てられたスタック領域は，関数の呼び出しからリターンした時に呼び出し元関数で解放されます。したがって，SP は呼び出された関数の入口と出口でリターン・アドレス格納領域を指します。

(3) スタック・フレームの構成
関数 f（Callee 側関数）が，関数 g（Caller 側関数）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，または設定できる領域の内容は以下です。

(a) 関数 f が受け取る仮引数を割り付ける領域
レジスタに割り付けない仮引数を設定する領域です。この領域に設定されたデータは 2 バイト境界にアラインされます。すべての仮引数をレジスタに割り付けるとこの領域のサイズが 0 となります。

(b) リターン・アドレス格納領域
リターン・アドレスを割り付ける領域です。領域のサイズは 4 バイト固定で，アドレスは far アドレスとして設定されます。
4 バイトのうちの上位 1 バイトの値は不定です。

(c) 関数 f のローカル変数格納領域
関数 f がローカル変数を格納するために使用するスタック領域です。

(d) 関数 f から呼び出す関数に引き渡す実引数を割り付ける領域
関数 f がほかの関数を呼び出す際に，スタックに割り付ける実引数を設定する領域です。呼び出しに要する実引数を全てレジスタに割り付けるなら，この領域のサイズは 0 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-RL 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 9.2 識別子について

C	アセンブリ・プログラム
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 “_” を取り除いた名前で参照してください。

(2) スタック・フレームに関して
CC-RL 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 言語関数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，その際は関数の入口と出口で SP の値が同じにな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は，自由に使う事ができます（使用前にレジスタの値を退避し，使用後に復帰する必要はありません）。

(3) C 言語関数への戻り先アドレス
CC-RL は「関数の戻り先アドレスは “ スタック先頭 ” 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，スタック先頭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 “ret” を実行してください。

(4) アセンブラ関数のプロトタイプ宣言
CC-RL では C 言語関数から呼び出す関数に対してプロトタイプ宣言が必要です。アセンブラ関数であっても正しいプロトタイプ宣言をしてください。

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-RL 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) レジスタ
CC-RL は C 言語関数呼び出しの前後において，一部のレジスタを除き，レジスタの値が同一であることを保証しません 。そのため，保持しなくてはならない値をレジスタに割り当てたままにしないでください。

(3) アセンブラ関数への戻り先アドレス
CC-RL は「関数の戻り先アドレスは “ スタック先頭 ” 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，スタック先頭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，call 命令を使用して分岐することにより，関数の戻り先アドレスがスタック先頭に格納されます。

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-RL アセンブラでは，次のように行います。

```
        .PUBLIC _i
        .PUBLIC _c
        .SECTION        .data, DATA
_i:
        .DB4    0x0
_c:
        .DB     0x0
```

10. メッセージ

ここでは，CC-RL 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-RL が出力する内部エラー・メッセージ，エラー・メッセージ，致命的エラー・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-RL 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	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	致命的エラー：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-RL が出力するメッセージ番号について説明します。
CC-RL 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-RL 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（*情報*）。
	［対処方法］	特約店，または当社までご連絡ください。
C0511200	［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。
C0519996	［メッセージ］	Out of memory.
	［説明］	ccrl コマンドへの入力（ソース・ファイル名や指定オプション）の量が多すぎます。
	［対処方法］	ccrl コマンドへの入力を分割して，複数回に分けて起動してください。
C0519997	［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。
C0520000	［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。
C0529000	［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。
C0550802	［メッセージ］	Internal error(action type of icode strage).
	［対処方法］	特約店，または当社までご連絡ください。
C0550804	［メッセージ］	Internal error(section name ptr not found(*文字列*)).
	［対処方法］	特約店，または当社までご連絡ください。
C0550805	［メッセージ］	Internal error(section list ptr not found(*文字列*)).
	［対処方法］	特約店，または当社までご連絡ください。
C0550806	［メッセージ］	Internal error(current section ptr not found(*文字列*)).
	［対処方法］	特約店，または当社までご連絡ください。

C0550808	［メッセージ］	Internal error(*文字列*).
	［対処方法］	特約店，または当社までご連絡ください。
C0551800	［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。
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。
C0564001	［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。
C0580013	［メッセージ］	internal error. unexpected syntax error message.
	［対処方法］	特約店または弊社までご連絡ください。
C0580014	［メッセージ］	internal error. section "%s" is not found.
	［対処方法］	特約店または弊社までご連絡ください。
C0580015	［メッセージ］	internal error. function for "%s" is not found.
	［対処方法］	特約店または弊社までご連絡ください。
C0580016	［メッセージ］	internal error. additional bytes are invalid "%s".
	［対処方法］	特約店または弊社までご連絡ください。
C0580900	［メッセージ］	internal error. undefined message for "%s".
	［対処方法］	特約店または弊社までご連絡ください。
C0580901	［メッセージ］	internal error. failed to set signal handler.
	［対処方法］	特約店または弊社までご連絡ください。
C0580902	［メッセージ］	internal error. invalid memory access.
	［対処方法］	特約店または弊社までご連絡ください。
C0580903	［メッセージ］	internal error. invalid instruction execution.
	［対処方法］	特約店または弊社までご連絡ください。
C0580904	［メッセージ］	internal error. abnormal termination.
	［対処方法］	特約店または弊社までご連絡ください。
C0580905	［メッセージ］	internal error. illegal operation.
	［対処方法］	特約店または弊社までご連絡ください。
C0580906	［メッセージ］	internal error. unexpected signal received.
	［対処方法］	特約店または弊社までご連絡ください。
C0580907	［メッセージ］	internal error. unknown message type "%s".
	［対処方法］	特約店または弊社までご連絡ください。
C0590001	［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。

10.5.2 エラー

表 10.3 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。
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。
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。
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。
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。
E0511108	[メッセージ]	" *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。
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。
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。
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。
E0511115	[メッセージ]	"-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。
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。
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。
E0511124	[メッセージ]	"-cpu" オプションを指定してください。
E0511129	[メッセージ]	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]	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]	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。
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。
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。
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。
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。
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。
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。
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。
E0511177	[メッセージ]	" *文字列* " が複数回指定されています。
E0511178	[メッセージ]	*バージョン*に対応した professional 版のライセンスが確認できませんでした。" *文字列* " オプションを使用できません。professional 版の購入を検討ください。
E0511182	[メッセージ]	ファイルアクセスエラー。(*情報*)

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。
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。
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。
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。
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。
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。
E0520011	［メッセージ］	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。
E0520013	［メッセージ］	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。
	［対処方法］	統合開発環境でプロジェクトを作成した時に iodefine.h ファイルが生成されます。この中で割り込み要求名が定義されていますので，割り込み要求名を記述している場合は，#pragma 指令の前に iodefine.h をインクルードしてください。なお，ファイルのインクルードは，コンパイラの -preinclude オプションでも指定可能です。
E0520017	［メッセージ］	"]" がありません。
E0520018	［メッセージ］	")" 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。
E0520020	［メッセージ］	識別子 " *文字列* " は定義されていません。
	［対処方法］	統合開発環境でプロジェクトを作成した時に iodefine.h ファイルが生成されます。SFR の予約語を使用する際には，このファイルをインクルードしてください。 PSW を直接アクセスすることはできません。PSW を操作する場合は，組み込み関数 __get_psw または __set_psw を用いてください。ビットアクセスする場合は，iodefine.h で定義されている型を利用してアクセスしてください。
E0520022	［メッセージ］	不正な 16 進数です。
E0520023	［メッセージ］	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。
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。
E0520026	［メッセージ］	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。
E0520028	［メッセージ］	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。
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。

E0520031	［メッセージ］	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。
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。
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。
E0520036	［メッセージ］	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。
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。
E0520039	［メッセージ］	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。
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。
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*")。
E0520044	［メッセージ］	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。
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。
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。
E0520049	［メッセージ］	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。
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。
E0520052	［メッセージ］	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。
E0520054	［メッセージ］	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。
E0520056	［メッセージ］	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。
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。
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。
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。
E0520061	［メッセージ］	整数演算の結果が範囲を超えました。
E0520062	［メッセージ］	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。
E0520064	［メッセージ］	宣言は何も宣言できません。

E0520065	［メッセージ］	";" がありません。
	［対処方法］	統合開発環境でプロジェクトを作成した時に iodefine.h ファイルが生成されます。 SFR の予約語を使用する際には，このファイルをインクルードしてください。 PSW を直接アクセスすることはできません。PSW を操作する場合は，組み込み関数 __get_psw または __set_psw を用いてください。ビットアクセスする場合は，iodefine.h で定義されている型を利用してアクセスしてください。
E0520066	［メッセージ］	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。
E0520069	［メッセージ］	整数変換で結果の値が丸められました。
E0520070	［メッセージ］	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。
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。
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。
E0520078	［メッセージ］	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。
E0520080	［メッセージ］	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。
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。
E0520084	［メッセージ］	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。
E0520086	［メッセージ］	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。
E0520088	［メッセージ］	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。
E0520090	［メッセージ］	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。
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。
E0520093	［メッセージ］	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。
E0520095	［メッセージ］	配列が大きすぎます。
E0520097	［メッセージ］	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。
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。
E0520100	［メッセージ］	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。
E0520102	［メッセージ］	列挙型の前方宣言は標準でありません。

E0520104	[メッセージ]	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。
E0520106	[メッセージ]	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。
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。
E0520110	[メッセージ]	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。
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。
E0520115	[メッセージ]	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。
E0520117	[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。
E0520118	[メッセージ]	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。
E0520120	[メッセージ]	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。
E0520122	[メッセージ]	default ラベルは switch の中でのみ使用できます。
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。
E0520125	[メッセージ]	"(" がありません。
E0520127	[メッセージ]	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。
E0520130	[メッセージ]	"{" がありません。
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。
E0520134	[メッセージ]	フィールド名がありません。
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。
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。
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。
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。
E0520140	[メッセージ]	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。
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。
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。
E0520145	[メッセージ]	*種別* " *シンボル名* " 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。
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。
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。
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。

E0520151	[メッセージ]	typedef 名は引数として再宣言できません。
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。
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。
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。
E0520165	[メッセージ]	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。
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。
E0520168	[メッセージ]	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。
E0520171	[メッセージ]	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。
E0520175	[メッセージ]	添字が範囲を超えました。
E0520179	[メッセージ]	"%" 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。
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。
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。
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。
E0520223	[メッセージ]	関数 *関数名* は暗黙に宣言されました。
E0520228	[メッセージ]	最後のカンマは標準ではありません。
	[説明]	カンマが仕様にあっていません。
E0520235	[メッセージ]	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。
E0520240	[メッセージ]	宣言の指示子が重複しています。
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。
E0520253	[メッセージ]	"," がありません。
E0520254	[メッセージ]	型名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。
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。
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。
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。
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。指定されたとおりに式を評価します。
E0520296	[メッセージ]	左辺値でない配列の不正な利用です。

E0520301	［メッセージ］	*typedef* 名はすでに同じ型で宣言されています。
E0520313	［メッセージ］	型修飾子はこの関数に許されていません。
E0520325	［メッセージ］	inline 指定子は関数宣言のみに利用できます。
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。
E0520375	［メッセージ］	宣言は typedef 名を必要とします。
E0520393	［メッセージ］	不完全クラス型へのポインタ型は許されていません。
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。
E0520404	［メッセージ］	関数 "main" は inline 宣言できません。
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。
E0520411	［メッセージ］	引数は許されていません。
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。
E0520450	［メッセージ］	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。
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。
E0520469	［メッセージ］	*xxx* のタグの種類は，*シンボル* の宣言と一致しません。
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。
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。
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。
E0520520	［メッセージ］	集成体は "{...}" により初期化してください。
E0520521	［メッセージ］	メンバへのポインタが選択したクラスは適合しません (型 1 と 型 2)。
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。
E0520526	［メッセージ］	引数は void 型を持てません。
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。
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 :

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。
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。
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。
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。
E0520643	［メッセージ］	"restrict" は許されていません。
E0520644	［メッセージ］	関数へのポインタまたは参照は，"restrict" 修飾できません。
E0520654	［メッセージ］	宣言修飾子が前の宣言と合致しません。
E0520655	［メッセージ］	修飾子 *名前* はこの宣言では許されていません。
E0520660	［メッセージ］	パッキング値が不正です。
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。
E0520702	［メッセージ］	"=" がありません。
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。
E0520731	［メッセージ］	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。
E0520744	［メッセージ］	適合しないメモリアトリビュートが指定されました。
E0520747	［メッセージ］	メモリアトリビュートが複数回指定されています。
E0520749	［メッセージ］	型指定子は許されていません。
E0520757	［メッセージ］	*名前* は型名ではありません。
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］	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。
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。
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。
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。
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。
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。
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。
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。
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。
E0520862	［メッセージ］	関数は不完全型 " *型名* " を返します。
E0520870	［メッセージ］	不正な多バイト文字列です。
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。
E0520935	［メッセージ］	"typedef" はここでは指定できません。
E0520938	［メッセージ］	返却型 "int" が関数 "main" の宣言で省略されました。
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。
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。

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。
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。
E0520965	［メッセージ］	間違ったユニバーサル・キャラクタ名です。
E0520966	［メッセージ］	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］	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］	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］	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。
E0520976	［メッセージ］	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］	複合リテラル型 *名前* は許されていません。
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。
E0520992	［メッセージ］	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。
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。
E0521029	［メッセージ］	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］	静的変数は inline 関数で定義できません。
E0521031	［メッセージ］	内部リンケージを持つ実体は外部リンケージを持つ inline 関数で参照できません。
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。
E0521037	［メッセージ］	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］	文字列リテラルがありません。
E0521039	［メッセージ］	認識されない STDC pragma です。
E0521040	［メッセージ］	"ON" か "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。
E0521045	［メッセージ］	指示子の種類が不正です。
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］	可変長の配列に初期化子は指定できません。

E0521051	［メッセージ］	*名前* 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］	インライン関数 *名前* には定義が必要です。
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。
E0521062	［メッセージ］	The other declaration is %p.
	［説明］	別の宣言があります。
E0521072	［メッセージ］	宣言はラベルを持てません。
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］	記憶域クラスは auto または register である必要があります。
E0521158	［メッセージ］	返却型 void は修飾できません。
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。
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。
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。
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。
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。
E0521348	［メッセージ］	Declaration hides "*symbol*".
	［説明］	宣言は "*symbol*" を隠します。
E0521352	［メッセージ］	Expected "SAT" or "DEFAULT".
	［説明］	"SAT" か "DEFAULT" がありません。
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。
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。

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。
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。
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。
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。
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。
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。
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。
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。
E0521633	［メッセージ］	セクション名が指定されていません。

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。
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］	*名前* とその置換テキストの間には空白が必要です。
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。
E0523003	［メッセージ］	__sectop/__secend/__secsize にセクション名がありません。
E0523004	［メッセージ］	セクション名がありません。
	［説明］	セクション名を表す文字列がない，または使用不可能な文字を使っています。
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。
	［対処方法］	統合開発環境でプロジェクトを作成した時に iodefine.h ファイルが生成されます。この中で，割り込み要求名が定義されていますので，割り込み要求名を使用する C ソースファイルでは，iodefine.h をインクルードしてください。 また，CA78K0R と CC-RL とで割り込み関数の記述形式が異なります。CC-RL では移行支援機能のオプションを用意していますので，コンパイラの -convert_cc オプションを使用すると，一部の記述については CA78K0R の形式で記述することが可能です。
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。
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。
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。
E0523014	［メッセージ］	不正な 2 進数です。

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。
E0523038	［メッセージ］	ひとつの構造体 / 共用体 / クラスの中に，異なる pack 値を持つものが混在しています。
E0523044	［メッセージ］	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。
E0523048	［メッセージ］	割り込み関数を不正に参照しています。
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］	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。
	［説明］	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。
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。
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。
E0523077	［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。
E0523078	［メッセージ］	*xxx* は CC-RL で使えません。
E0523087	［メッセージ］	" *関数名* " を不正に参照しています。
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。
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。
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。
E0550212	［メッセージ］	指定されたシンボルはすでに *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。
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。
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］	（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。
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。
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。
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実引数が不正です。
	［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実引数が不正です。
	［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。
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。
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。
E0550271	［メッセージ］	" *文字列 1*" は先に指定された " *文字列 2*" と矛盾します
	［説明］	" *文字列 1*" が先に指定された " *文字列 2*" と矛盾しています。ソースの記述を確認してください。 補足）"align=0" は，.section 疑似命令に align 指定がない場合を意味します。
E0550272	［メッセージ］	" *文字列*" が必要です。
	［対処方法］	該当行に " *文字列*" の指定を追加してください。
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。
E0550605	[メッセージ]	" *文字列 1*" オプションで指定された " *文字列 2*" はファイルです。フォルダを指定してください。
E0550606	[メッセージ]	" *文字列 1*" オプションで指定されたフォルダ " *文字列 2*" 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。
E0550624	[メッセージ]	"-cpu" 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。
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。
E0550633	[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。
E0550637	[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。
E0550638	[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。
E0550639	[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。
E0550640	[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。
E0550641	[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。
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。
E0550647	[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。
E0550649	[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。
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。

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。
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。
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。
	［説明］	不適切な SFR，または制御レジスタが記述されています。
E0551317	［メッセージ］	SFR は前方参照できません。
E0551401	［メッセージ］	"*operand*" の記述に誤りがあります。
	［説明］	オペランドの記述に誤りがあります。
	［対処方法］	オペランドの記述を確認してください。
E0551402	［説明］	命令の種類に誤りがあります。
E0551403	［メッセージ］	.DB8 疑似命令のオペランドには記述できません。
	［説明］	.DB8 疑似命令のオペランドには分離演算子を記述できません。
	［対処方法］	.DB8 のオペランドが正しく記述されているか確認してください。
E0551404	［メッセージ］	ベクタ・テーブル・アドレスの指定に誤りがあります。
	［対処方法］	.VECTOR 疑似命令のアドレスが正しく指定されているか確認してください。
E0551405	［メッセージ］	$MIRROR 指定できないシンボルです。
	［対処方法］	$MIRROR 疑似命令で外部参照名以外を指定していないか確認してください。
E0551406	［メッセージ］	" *疑似命令* " に "." で始まるシンボルを指定できません。
E0551501	［メッセージ］	"-output" オプションを指定したとき，複数ソース・ファイルを入力できません。
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。
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。
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。
E0562114	［メッセージ］	The generated duplicate section name "*section*" is confused
	［説明］	同名セクション *section* が複数回現れ，処理できませんでした。
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］	【V1.06 以前】 Interrupt number " *ベクタテーブル・アドレス* " has invalid interrupt jump address : " *シンボル* " 【V1.07 以降】 Interrupt table address " *ベクタテーブル・アドレス* " 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］	【V1.06 以前】 Interrupt number " *ベクタテーブル・アドレス* " of " *セクション* " has multiple definition 【V1.07 以降】 Interrupt table address " *ベクタテーブル・アドレス* " 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。
	［対処方法］	CA78K0R と CC-RL はオブジェクトファイルの形式が異なるため，CA78K0R で作成したオブジェクトファイルを CC-RL でリンクすることはできません。CA78K0R のオブジェクトファイルを指定している場合は，CC-RL 用にオブジェクトファイルを作成しなおして，リンクしてください。
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。
	［対処方法］	CA78K0R のライブラリファイルを指定していないか確認してください。CA78K0R と CC-RL はオブジェクトファイルの形式が異なるため，CA78K0R で作成したライブラリを CC-RL でリンクすることはできません。CC-RL 用にライブラリファイルを作成しなおして，リンクしてください。
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。
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。

E0562225	［メッセージ］	Device file mismatch " *デバイス・ファイル* " in " *入力ファイル* "
	［説明］	異なるデバイス・ファイルを使用して作成したオブジェクト・ファイルをリンクしようとしています。 または，オブジェクト・ファイル作成時に使用したデバイス・ファイルと -device オプションで指定したデバイス・ファイルが異なっています。
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。
	［対処方法］	セクションの配置アドレスとサイズを確認してください。
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。
E0562325	［メッセージ］	Section " *セクション* " steps over the border of " *境界* "
	［説明］	" *セクション* " が " *境界* " をまたいで配置されています。
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。
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。
E0562350	［メッセージ］	Section " *セクション* " cannot be placed on the " *領域* ".
	［説明］	-self オプション指定時，" *領域* " に " *セクション* " を配置することはできません。
E0562351	［メッセージ］	Section " *セクション* " cannot be placed on the " *領域* ".
	［説明］	-ocdtr オプション指定時，" *領域* " に " *セクション* " を配置することはできません。

E0562352	［メッセージ］	Section " *セクション* " cannot be placed on the " *領域* ".
	［説明］	-ocdhpi オプション指定時，" *領域* " に " *セクション* " を配置することはできません。
E0562353	［メッセージ］	Section " *セクション* " address overflowed out of range " *領域* ".
	［説明］	-selfw オプション指定時，" *領域* " をまたいで " *セクション* " を配置することはできません。
E0562354	［メッセージ］	Section " *セクション* " address overflowed out of range " *領域* ".
	［説明］	-ocdtrw オプション指定時，" *領域* " をまたいで " *セクション* " を配置することはできません。
E0562355	［メッセージ］	Section " *セクション* " address overflowed out of range " *領域* ".
	［説明］	-ocdhpiw オプション指定時，" *領域* " をまたいで " *セクション* " を配置することはできません。
E0562360	［メッセージ］	CRC result cannot be placed on the " *領域* ".
	［説明］	-self オプション指定時，" *領域* " に CRC 演算結果を配置することはできません。
E0562361	［メッセージ］	CRC result cannot be placed on the " *領域* ".
	［説明］	-ocdtr オプション指定時，" *領域* " に CRC 演算結果を配置することはできません。
E0562362	［メッセージ］	CRC result cannot be placed on the " *領域* ".
	［説明］	-ocdhpi オプション指定時，" *領域* " に CRC 演算結果を配置することはできません。
E0562363	［メッセージ］	CRC result address overflowed out of range " *領域* ".
	［説明］	-selfw オプション指定時，" *領域* " をまたいで CRC 演算結果を配置することはできません。
E0562364	［メッセージ］	CRC result address overflowed out of range " *領域* ".
	［説明］	-ocdtrw オプション指定時，" *領域* " をまたいで CRC 演算結果を配置することはできません。
E0562365	［メッセージ］	CRC result address overflowed out of range " *領域* ".
	［説明］	-ocdhpiw オプション指定時，" *領域* " をまたいで CRC 演算結果を配置することはできません。
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。
E0562600	［メッセージ］	Library " *ライブラリ* " requires " *ライセンス・エディション* "
	［説明］	指定した " *ライブラリ* " の利用には，" *ライセンス・エディション* " が必要です。
E0580001	［メッセージ］	"SMSG%s" cannot be specified as dst.
	［説明］	SMSG0，SMSG15 には値を書き込めません。
	［対処方法］	SMSG0，SMSG15 以外を指定してください。
E0580002	［メッセージ］	"%s" is out of range (%d-%d).
	［説明］	オペランドの値が指定可能な範囲ではありません。
	［対処方法］	オペランドの値を確認してください。
E0580003	［メッセージ］	label cannot be specified as memory operand.
	［説明］	ラベルはメモリオペランドのオフセットに記述できません。
	［対処方法］	オペランドの記述を確認してください。
E0580004	［メッセージ］	branch destination is out of program area.
	［説明］	分岐命令の分岐先がプログラムの外です。
	［対処方法］	分岐命令の分岐距離を確認してください。
E0580005	［メッセージ］	invalid address format.
	［対処方法］	$addr には数値またはラベルのみを指定してください。

E0580006	［メッセージ］	unknown label "%s".
	［説明］	分岐命令の分岐先ラベルがみつかりません。
	［対処方法］	オペランドの記述を確認してください。
E0580007	［メッセージ］	expecting "%s".
	［説明］	構文に誤りがあります。
	［対処方法］	記述を確認してください。
E0580008	［メッセージ］	unexpected "%s".
	［説明］	構文に誤りがあります。
	［対処方法］	記述を確認してください。
E0580009	［メッセージ］	"%s" is already defined.
	［説明］	ラベルが多重に定義されています。
	［対処方法］	ラベル名を確認してください。
E0580010	［メッセージ］	illegal operation ("%s").
	［説明］	式の記述に誤りがあります。
	［対処方法］	記述を確認してください。
E0580011	［メッセージ］	machine instructions cannot exceed 32 instructions.
	［説明］	プログラムの命令数が 32 個より多いです。
	［対処方法］	プログラムを縮小してください。
E0580012	［メッセージ］	illegal symbol ("%s").
	［説明］	シンボル名に不正な文字が使われています。
	［対処方法］	シンボル名を確認してください。
E0580017	［メッセージ］	suffix and prefix cannot be specified together.
	［説明］	16 進数の記述に誤りがあります。
	［対処方法］	16 進数値のプレフィックスサフィックスはどちらか片方のみを記述してください。
E0580100	［メッセージ］	specify .SECTION directive.
	［対処方法］	プログラムの先頭には .SECTION 疑似命令を記述してください。
E0580104	［メッセージ］	"%s" is already defined.
	［説明］	同名のマクロが複数あります。
	［対処方法］	マクロ名を確認してください。
E0580105	［メッセージ］	illegal macro argument.
	［説明］	構文に誤りがあります。
	［対処方法］	記述を確認してください。
E0580106	［メッセージ］	actual argument of macro is not matched.
	［説明］	マクロ参照の実パラメータの数が仮パラメータの数と一致していません。
	［対処方法］	記述を確認してください。

E0580200	［メッセージ］	illegal syntax.
	［説明］	構文に誤りがあります。
	［対処方法］	記述を確認してください。
E0580201	［メッセージ］	include file cannot nest over 4294967294 times.
	［説明］	$include 制御命令の入れ子の回数が上限を超えました。
E0580202	［メッセージ］	condition assembly directive cannot nest over 4294967294 times.
	［説明］	$ifdef 制御命令の入れ子の回数が上限を超えました。
E0580203	［メッセージ］	unexpected directive "%s".
	［説明］	"%s" に対応する $if,$ifdef 制御命令がありません。
E0580204	［メッセージ］	expecting directive "%s".
	［説明］	"%s" に対応する $endif 制御命令がありません。
E0580300	［メッセージ］	"%s" is unrecognized.
	［説明］	不正なオプション名です。
	［対処方法］	オプション指定を確認してください。
E0580301	［メッセージ］	specify the "-o" option.
	［対処方法］	-o オプションを指定してください。
E0580302	［メッセージ］	"%s" option requires an argument.
	［対処方法］	オプションにパラメータを指定してください。
E0580303	［メッセージ］	invalid argument for the "%s" option.
	［説明］	オプションのパラメータ指定が不正です。
	［対処方法］	オプション指定を確認してください。
E0580305	［メッセージ］	"%s" specified in the "%s" option is invalid name.
	［説明］	-D または -U オプションで指定したシンボル名が不正です。
	［対処方法］	-D または -U オプションの指定を確認してください。
E0580307	［メッセージ］	cannot open subcommand file "%s".
	［説明］	サブコマンド・ファイルを開くことができません。
	［対処方法］	サブコマンド・ファイルの状態を確認してください。
E0580308	［メッセージ］	cannot read subcommand file "%s".
	［説明］	サブコマンド・ファイルを読むことができません。
	［対処方法］	サブコマンド・ファイルの状態を確認してください。
E0580309	［メッセージ］	subcommand file "%s" is read more than once.
	［説明］	同名のサブコマンド・ファイルが複数回指定されています。
	［対処方法］	サブコマンド・ファイル指定を確認してください。
E0580310	［メッセージ］	missing double quotation.
	［説明］	2 重引用符が欠けています。
	［対処方法］	記述を確認してください。

E0580311	［メッセージ］	specify the input file.
	［対処方法］	入力ファイルを 1 つ指定してください。
E0580312	［メッセージ］	too many input files.
	［対処方法］	入力ファイル指定は 1 つのみにしてください。
E0580313	［メッセージ］	the file "%s" specified by the "%s" option is a folder.
	［説明］	指定された出力先ファイルはフォルダです。
	［対処方法］	出力ファイルオプションの指定を確認してください。
E0580314	［メッセージ］	the output folder "%s" specified by the "%s" option is not found.
	［説明］	出力先フォルダがみつかりません。
	［対処方法］	出力ファイルオプションの指定を確認してください。
E0580315	［メッセージ］	cannot open input file "%s".
	［説明］	入力ファイルを開くことができません。
	［対処方法］	入力ファイルの状態を確認してください。
E0580316	［メッセージ］	cannot read input file "%s".
	［説明］	入力ファイルを読むことができません。
	［対処方法］	入力ファイルのを確認してください。
E0580317	［メッセージ］	cannot open output file "%s".
	［説明］	出力先ファイルを開くことができません。
	［対処方法］	出力先ファイルの状態を確認してください。
E0580318	［メッセージ］	cannot write output file "%s".
	［説明］	出力先ファイルへの書き込みに失敗しました。
	［対処方法］	出力先ファイルの状態を確認してください。

10.5.3 致命的エラー

表 10.4 致命的エラー

F0520003	［メッセージ］	#include ファイル " *ファイル名* " は自分自身をインクルードしています。
	［説明］	#include ファイル " *ファイル名* " は自分自身をインクルードしています。修正してください。
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。
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。
F0520013	［メッセージ］	ファイル名がありません。
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。
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。
F0520163	［メッセージ］	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］	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］	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。
F0520563	［メッセージ］	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。
F0520571	［メッセージ］	不正なオプションです。: *オプション名*
F0520642	［メッセージ］	テンポラリ・ファイル名を生成できません。
F0520920	［メッセージ］	出力ファイルがオープンできません : *xxx*。
F0523029	［メッセージ］	misra ルールファイルをオープンできません。
	［説明］	-misra2004=" *ファイル名* " または -misra2012=" *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］	-misra2004=" *ファイル名* " または -misra2012=" *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。
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。
F0523088	［メッセージ］	ビット位置が範囲外の値です。
F0530320	［メッセージ］	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。
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。
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。

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。
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。
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。
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。
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。
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。
F0540204	［メッセージ］	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］	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。
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。
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。
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。
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。
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］	条件アセンブル疑似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］	条件アセンブル疑似命令を確認してください。
F0550512	［メッセージ］	条件アセンブル疑似命令が 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。
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。
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。
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。
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。
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越えました。
	［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*) とセクション (*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・エントリに出力するシンボルが限界数 (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。
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。
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。
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。
F0563003	［メッセージ］	Illegal file format " *ファイル* "
	［説明］	" *ファイル* " が利用できないファイル形式です。
F0563004	［メッセージ］	Invalid inter-module optimization information type 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。
F0563006	［メッセージ］	Option " *オプション* " cannot be combined with library
	［説明］	" *オプション* " とコンパイラで作成したライブラリは同時に指定できません。ライブラリファイルとオプシ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］	バイナリ・ファイルは -binary オプションで指定し，共にリンクする .obj/.rel ファイルがあること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*" in " *ファイル*"
	［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。
F0563103	［メッセージ］	Section size overflow : " *セクション*"
	［説明］	" *セクション*" が使用可能なサイズを超えました。
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。 CS+ で RL78-S2 コアのマイコンのプロジェクトを作成した場合には，乗除積和演算器を使用する設定でプロジェクトが生成されます。CS+ で乗除積和演算器を搭載していないマイコンでプロジェクトを作成する場合は，ビルドツールのプロパティの共通オプションタブを表示し，その中の「演算器を使用する」の設定を「使用しない（-use_mda=not_use）」にしてください。
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。
F0563115	［メッセージ］	Cpu type in " *ファイル*" is not supported
	［説明］	" *ファイル*" の CPU 種別に対応していません。入力ファイルが正しいか確認してください。
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* " in " *ファイル* "
	［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。
F0563150	［メッセージ］	Multiple files cannot be specified while processing " *プロセス* "
	［説明］	" *プロセス* " 処理に対して，複数のファイルを指定できません。
	［対処方法］	ファイルの指定を確認してください。
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。
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。
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。
F0563430	［メッセージ］	The total section size exceeded the limit of the evaluation version of *バージョン*. Please consider purchasing the product.
	［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。 リンク・サイズを 64K バイト以内に制限しています。製品版の購入をご検討ください。【V1.11 以前のみ】
F0563431	［メッセージ］	Incorrect device type, object file mismatch.
	［説明］	対応しない CPU 種別を入力しました。
	［対処方法］	リンカ実行ファイルとオプションのファイルを確認してください。
F0563600	［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。
F0563601	［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。
F0563602	［メッセージ］	" *文字列* " option requires "*edition*".
	［説明］	" *文字列* " オプションの使用には *edition* が必要です。

F0580101	［メッセージ］	section name is not specified in the .SECTION directive.
	［対処方法］	.SECTION 疑似命令にはセクション名を指定してください。
F0580102	［メッセージ］	"%s" is already specified.
	［説明］	ファイル内に同名のセクションがあります。
	［対処方法］	セクション名を確認してください。
F0580103	［メッセージ］	cannot specify a section name in the .PSECTION directive.
	［対処方法］	.PSECTION 疑似命令にはセクション名を記述しないでください。
F0580399	［メッセージ］	too many errors.
	［説明］	エラーの数が多すぎるため，処理を中断しました。
F0593000	［メッセージ］	‘-cpu’ option is specified twice
	［説明］	-cpu オプションが複数回指定されました。有効な指定を確認してください。
F0593021	［メッセージ］	Memory overflow
	［説明］	メモリが不足しています。他のアプリケーションを終了して，再度ライブラリを生成し直してください。
F0593300	［メッセージ］	Cannot open internal file
	［説明］	内部ファイルをオープンできませんでした。
F0593302	［メッセージ］	Cannot input internal file
	［説明］	内部ファイルの読み込みに失敗しました。
F0593303	［メッセージ］	Cannot output internal file
	［説明］	内部ファイルへの書き込みに失敗しました。
F0593305	［メッセージ］	Invalid command parameter " *オプション名* "
	［説明］	" *オプション名* " の指定に誤りがあります。
F0593320	［メッセージ］	Command parameter buffer overflow
	［説明］	内部バッファが不足しています。
F0593321	［メッセージ］	Illegal environment variable
	［説明］	環境設定の指定に誤りがあります。設定を見直してください。
F0593322	［メッセージ］	Lacking cpu specification
	［説明］	-cpu オプションが指定されていません。設定を確認してください。
F0593324	［メッセージ］	Cannot open subcommand file " *サブコマンド・ファイル名* "
	［説明］	" *サブコマンド・ファイル名* " をオープンできませんでした。
F0593325	［メッセージ］	Cannot close subcommand file
	［説明］	サブコマンド・ファイルをクローズできませんでした。
F0593326	［メッセージ］	Cannot input subcommand file
	［説明］	サブコマンド・ファイルの読み込みに失敗しました。
F0593327	［メッセージ］	Cannot get compiler version
	［説明］	コンパイラのバージョンを取得できませんでした。

F0593328	［メッセージ］	Cannot find archive file
	［説明］	CC-RL を構成するファイルがみつかりません。再インストールしてください。
F0593329	［メッセージ］	Cannot find compiler program
	［説明］	CC-RL を構成するファイルがみつかりません。再インストールしてください。
F0593330	［メッセージ］	The " *オプション名 1*" option and the " *オプション名 2*" option are inconsistent
	［説明］	" *オプション名 1*" と " *オプション名 2*" の指定が矛盾しています。指定を確認してください。

10.5.4 インフォメーション

表 10.5 インフォメーション

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。
M0523086	［メッセージ］	Rule *ルール番号*: *内容*
	［説明］	MISRA-C:2012 の*ルール番号*と*内容*の該当箇所を検出しました。
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。
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。
M0560700	［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アドレス範囲を超えました。

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。
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。
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。
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。
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。
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。
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。
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。
W0511164	［メッセージ］	同じファイル名 " *ファイル名* " が複数指定されています。
W0511180	［メッセージ］	*バージョン*の評価期間の有効期限が切れています。
W0511181	［メッセージ］	内部情報ファイルにエラーがあります。(*情報*)
W0511183	［メッセージ］	ライセンスマネージャがインストールされていません。
	［説明］	ライセンス・マネージャがインストールされていません。対応するライセンス・マネージャをインストールしてください。
W0511184	［メッセージ］	" *文字列* " オプションが指定されたため，"-g" オプションが有効となります。
	［対処方法］	"-g" オプションを明示的に指定することにより，本メッセージを抑止することができます。
W0511185	［メッセージ］	professional 版の機能の試用期間は残り *数字* 日です。professional 版の購入を検討ください。
W0511186	［メッセージ］	" *バージョン* " のオプション " *文字列* " の評価期間は残り "*N*" 日です。評価期間経過後は，暗黙に "-Olite" 指定に変更されます。引き続き " *文字列* " を利用したい場合は製品の購入を検討ください。
	［説明］	全ての最適化レベルが使用可能な無償評価版として動作できる評価期間の残り日数を示しています。 評価期間後は使用可能な最適化レベルに制限が発生します。【V1.12 以降】 製品版の購入をご検討ください。
W0511187	［メッセージ］	" *バージョン* " のオプション " *文字列* " の評価期間の有効期限が切れています。暗黙に "-Olite" 指定に変更します。引き続き " *文字列* " を利用したい場合は製品の購入を検討ください。明示的に "-Olite" か "-Onothing" を指定することで，この警告は消えます。
	［説明］	無償評価版の評価期間が終了しています。使用可能な最適化レベルに制限が発生しています。【V1.12 以降】 製品版の購入をご検討ください。
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。
W0520011	［メッセージ］	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。

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。
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。
W0520027	［メッセージ］	char 型の値が範囲を超えています。
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。
W0520039	［メッセージ］	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*")。
W0520055	［メッセージ］	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。
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。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。
W0520064	［メッセージ］	この宣言は何も宣言できません。
W0520068	［メッセージ］	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。
W0520070	［メッセージ］	不完全型は許されていません。
W0520076	［メッセージ］	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。
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。
W0520083	［メッセージ］	型修飾子が複数回指定されました。
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。
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。
W0520111	［メッセージ］	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。
W0520127	［メッセージ］	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。
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。
W0520140	［メッセージ］	関数呼び出しに対する引数が多すぎます。
W0520152	［メッセージ］	0 でない値がポインタに変換されました。
W0520159	［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。
W0520161	［メッセージ］	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。
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。

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。
W0520171	[メッセージ]	不正な型変換です。
	[説明]	不正な型変換です。
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。
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。
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。
W0520175	[メッセージ]	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。
W0520179	[メッセージ]	"%" 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。
W0520186	[メッセージ]	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。
W0520188	[メッセージ]	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。
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。
W0520221	[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。
W0520222	[メッセージ]	浮動小数点演算の結果が範囲を超えました。
W0520223	[メッセージ]	関数 *関数名* は暗黙に宣言されました。
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。
W0520231	[メッセージ]	宣言は関数の外で見えません。
W0520236	[メッセージ]	制御式が定数です。
W0520240	[メッセージ]	宣言の指示子が重複しています。
W0520257	[メッセージ]	const " *シンボル名* " 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。
W0520301	[メッセージ]	typedef 名はすでに同じ型で宣言されています。
W0520375	[メッセージ]	宣言は typedef 名を必要とします。
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。
W0520513	[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。
W0520520	[メッセージ]	集成体は "{...}" により初期化してください。
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。
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。
W0520606	[メッセージ]	この pragma は宣言の直前でなければなりません。
W0520609	[メッセージ]	この種類の pragma はここでは使用できません。
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。

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。
W0520767	[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。
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。
W0520819	[メッセージ]	"..." は許されていません。
W0520867	[メッセージ]	"size_t" の宣言は期待された型 *型* 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。
W0520902	[メッセージ]	型修飾子を無視しました。
W0520940	[メッセージ]	void でない " *シンボル名* " に return 文がありません。
W0520951	[メッセージ]	main 関数の返却型は "int" である必要があります。
W0520966	[メッセージ]	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。
W0520967	[メッセージ]	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。
W0520968	[メッセージ]	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。
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。
W0521000	[メッセージ]	記憶域クラスはここでは指定できません。
W0521037	[メッセージ]	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。
W0521039	[メッセージ]	認識されない STDC pragma です。
W0521040	[メッセージ]	"ON" か "OFF" または "DEFAULT" がありません。
W0521046	[メッセージ]	浮動小数点の値が正しく記述されていません。
W0521051	[メッセージ]	*名前* 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。
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。
W0521057	[メッセージ]	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]	宣言はラベルを持てません。
W0521222	[メッセージ]	不正なエラー番号です。
W0521223	[メッセージ]	不正なエラータグです。
W0521224	[メッセージ]	エラー番号かエラータグがありません。
W0521297	[メッセージ]	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)。
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。
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。
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。
W0523018	[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。
W0523037	[メッセージ]	#pragma section 指定を無視します。
	[説明]	存在しないセクション種別を使っています。

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。
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。
W0523079	［メッセージ］	CC-RL に無い機能を使用しています。無視します。
	［説明］	移行支援機能ではサポートしていません。
W0523080	［メッセージ］	CC-RL のフォーマットで記述してください。
W0523081	［メッセージ］	CC-RL の機能に変換しました。
W0523082	［メッセージ］	偶数アライメントのオブジェクトを指すポインタが奇数番地を保持しています。
W0523083	［メッセージ］	奇数アドレスと型の組合せが不正です。
W0523084	［メッセージ］	“iodefine.h” をインクルードする必要があります。
W0523085	［メッセージ］	パッキングされた構造体メンバのアドレスを取得しています。
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。
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。
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。
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。
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。
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。
W0550019	［メッセージ］	オペランドに指定した値は*文字列*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*文字列*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。
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 : " *オプション* "
	［説明］	致命的エラー・メッセージはレベル変更できません。" *オプション* " 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。
W0561014	［メッセージ］	License manager is not installed
	［説明］	ライセンス・マネージャがインストールされていません。対応するライセンス・マネージャをインストールしてください。
W0561015	［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。"*パラメータ*" の指定を無視します。
W0561016	［メッセージ］	The evaluation version of *バージョン* is valid for the remaining *日数* days. After that, link size limit (64 Kbyte) will be applied. Please consider purchasing the product.
	［説明］	サイズ制限なし無償評価版として動作できる評価期間の残り日数を示しています。評価期間の後はサイズ制限あり無償評価版として動作します。【V1.11 以前のみ】製品版の購入をご検討ください。
W0561017	［メッセージ］	Paid license of "*バージョン*" is not found, and the evaluation period has expired. Please consider purchasing the product.
	［説明］	有償ライセンスが確認できません。また無償評価期間も終了しています。製品版の購入をご検討ください。
W0561018	［メッセージ］	The evaluation period of "*バージョン*" is valid for the remaining "*日数*" days. After that, functional limit will be applied. Please consider purchasing the product.
	［説明］	機能制限なし無償評価版として動作できる評価期間の残り日数を示しています。評価期間後は機能制限あり無償評価版として動作します。製品版の購入をご検討ください。
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 option "*オプション*" conflicts between symbol and section
	［説明］	"*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。

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。
W0561123	［メッセージ］	Section " *セクション*" is placed on the " *領域*"
	［説明］	-selfw オプション指定時，セクションを " *領域*" に配置しています。
W0561124	［メッセージ］	Section " *セクション*" is placed on the " *領域*"
	［説明］	-ocdtrw オプション指定時，セクションを " *領域*" に配置しています。
W0561125	［メッセージ］	Section " *セクション*" is placed on the " *領域*"
	［説明］	-ocdhpiw オプション指定時，セクションを " *領域*" に配置しています。
W0561126	［メッセージ］	CRC result is placed on the " *領域*"
	［説明］	-selfw オプション指定時，CRC 演算結果を " *領域*" に配置しています。
W0561127	［メッセージ］	CRC result is placed on the " *領域*"
	［説明］	-ocdtrw オプション指定時，CRC 演算結果を " *領域*" に配置しています。
W0561128	［メッセージ］	CRC result is placed on the " *領域*"
	［説明］	-ocdhpiw オプション指定時，CRC 演算結果を " *領域*" に配置しています。
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。
W0561142	［メッセージ］	Cannot find symbol which is a pair of " *シンボル* "
	［説明］	空き領域の範囲を表す " *シンボル* " と対になるシンボルが見つかりませんでした。
W0561143	［メッセージ］	Address *開始アドレス - 終了アドレス* cannot be placed on flash memory area.
	［説明］	*開始アドレス - 終了アドレス*の範囲はフラッシュメモリ領域になく，フラッシュライタで書き込みできないデータが存在します。
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。
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］	【V1.06 以前】 Interrupt number " *ベクタテーブル・アドレス* " of " *セクション* " is defined in input file 【V1.07 以降】 Interrupt table address " *ベクタテーブル・アドレス* " 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。
W0561184	［メッセージ］	【V1.06 以前】 Interrupt number " *ベクタテーブル・アドレス* " of " *セクション* " is defined 【V1.07 以降】 Interrupt table address " *ベクタテーブル・アドレス* " of " *セクション* " is defined
	［説明］	-debug_monitor オプションで記述したベクタ番号は，入力ファイル内，または -vectn オプションで定義済みです。 -debug_monitor オプションの指定を優先して，処理を継続します。
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。
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。
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・ポイント・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。
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。
W0561328	［メッセージ］	Section " *セクション* " border mismatch in " *ファイル* "
	［説明］	" *ファイル* " の " *セクション* " の配置境界属性が他と異なります。
W0561329	［メッセージ］	Devided option "start" at beginning of Section " *セクション* "
	［説明］	最適化によって 64K/64K-1 バイト境界をまたいでしまう可能性があるため，" *セクション* " 先頭で start オプション指定を分断しました。

W0561331	［メッセージ］	Section alignment is not adjusted : " *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。入力時の整列条件を満たしていない可能性があります。
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。
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］	Option " *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。
W0580304	［メッセージ］	"%s" option is specified more than once. The latter is valid.
	［説明］	同じオプションが複数回指定されています。
	［対処方法］	最後の指定が有効になります。
W0580306	［メッセージ］	the folder "%s" specified by the "-I" option is not found.
	［説明］	-I オプションで指定されたフォルダが見つかりません。
	［対処方法］	-I オプションの指定を確認してください。
W0591300	［メッセージ］	Command parameter specified twice " *オプション名* "
	［説明］	一度しか指定できないオプションが複数回指定されました。有効にしたいオプションを確認してください。
W0591301	［メッセージ］	" *オプション名* " option ignored
	［説明］	" *オプション名* " の指定を無視しました。

11. 注意事項

この章では，CC-RL を用いる際に注意すべき点について説明します。

11.1 コンパイラの注意事項

この節では，コンパイラの注意事項について説明します。

11.1.1 ポインタの間接参照

2 バイトの整列条件をもつ型へのポインタに奇数の値を設定して，そのポインタを間接参照すると不正動作となります。
コンパイラはポインタの間接参照に対して，そのポインタの指す型の整列条件に適合した命令を使用するためです。

例

```
int __near * volatile x;
int func(void){
        x = (int __near *)0xfaa1;
        return *x;                  // コンパイラは *x に対して ,2 バイトの整列条件に
                                    // 適合した命令を出力するため , コードの実行時に
                                    //*x は 0xfaa0 が指すデータを参照する
}
```

11.1.2 ポインタを介したレジスタ・アクセス

コンパイラの生成コード中で使用するレジスタを，ポインタを介してアクセスすると不正動作になることがあります。
コンパイラは，使用中のレジスタがポインタを介してアクセスされないことを前提としたコードを生成するためです。
以下のレジスタをポインタを介してアクセスする場合，プログラムの動作を保証しません。

- 使用する可能性のあるレジスタ・バンクに所属する汎用レジスタ
- SP
- PSW
- CS
- ES
- PMC

例

```
*(int __near *)0xfef8 = 7;      //0xffef8 は，バンク 0 の ax レジスタのアドレスなので，
                                // この記述を含み，かつバンク 0 を使用するプログラムの動作を
                                // 保証しない
```

11.1.3 関数呼び出し

あるファイルにおいてある関数を呼び出す時，そのファイル内にその関数の定義がない場合，またはその関数の定義を含むがその関数の定義よりも前にその関数を呼び出す場合，その関数の呼出しより前にその関数のプロトタイプを宣言することを推奨します。
関数呼出し時に仮引数の型がわからない時，コンパイラは関数定義における仮引数の型とは異なる型で関数を呼びだす可能性があり[注]，その場合，プログラムの実行結果が不正になるためです。

注 int 型や unsigned int 型より小さい算術型は int 型，またはそれで表現できない型は unsigned int 型に,float 型は double 型になります。
その他については，「(7) 既定の実引数拡張」を参照してください。

例 1. 不正な実行結果になるプログラム

```
AAA.c:
        void func(char a, char b);  //2 つの引数は各々 A，X レジスタから受け取る
BBB.c:
        extern void func();         // 引数の型が明示されていないのでプロトタイプ宣言ではない
        void main (void)
        {
                char    x=1, y=1;
                func(x, y);         //x と y は各々，int に拡張され，AX，BC レジスタに
        }                           // 割り付けられて関数呼び出しが行われる
```

例 2. 正しい実行結果になるプログラム

```
AAA.c:
        void func(char a, char b);          //2 つの引数は各々 A，X レジスタから受け取る
BBB.c:
        extern void func(char a, char b);   // プロトタイプ宣言
        void main (void)
        {
                char    x=1, y=1;
                func(x, y);                 //2 つの引数は各々 A，X レジスタに割り付けられて
        }                                   // 関数呼び出しが行われる
```

コンパイル・オプション “-refs_without_declaration” を指定すると関数の宣言がないことを検査できます。
プロトタイプが無い関数を呼び出すと，エラーとなります。

11.1.4 データ・フラッシュ領域

CC-RL はデータ・フラッシュ領域に対応したコード出力を行っていません。
アセンブラによる記述でアクセスしてください。
C ソースで記述する場合，8 ビット・データとしてアクセスしてください。

11.1.5 K&R 形式の関数定義 (_Bool 型の仮引数)

_Bool 型の仮引数を含む関数定義を K&R 形式で記述した場合，_Bool 型の仮引数に実引数の値をそのまま代入するアセンブラコードを生成します。このため，_Bool 型の実引数が 0,1 以外の場合は仮引数に 0 でも 1 でもない値を設定します。

例

```
void sub();
signed char c;
void func(void) {
        sub(2);
}

void sub(b)                         // 仮引数 b に 2 を設定
_Bool b;
{
        if (b == 0 || b == 1) {
                c = b;
        } else {
                c = -1;
        }
}
```

【回避策】
_Bool 型の引数を含む関数定義と関数宣言を関数原型形式で記述してください。

```
void sub(_Bool);                    // 関数原型
signed char c;
void func(void) {
        sub(2);
}

void sub(_Bool b)                   // 関数原型
{
        if (b == 0 || b == 1) {
                c = b;
        } else {
                c = -1;
        }
}
```

11.1.6 MISRA2004 チェック (ルール番号 10.1)

以下の条件をすべて満たすファイルに含まれる，列挙型の変数，返値又は列挙子を用いた文に対して不要なメッセージを出力することがあります。

- -signed_char オプションが指定されず，列挙子の値の範囲が 0 ～ +255 の範囲内にある列挙型の定義を含む
  または
  -signed_char オプションが指定され，列挙子の値の範囲が −128 ～ +255 の範囲内にある列挙型の定義を含む
- 列挙型の変数，返値又は列挙子を用いた文を含む
- MISRA のルール番号 10.1 をチェックするコンパイルオプションが指定される

例

```
typedef enum E { E1 = 0, E2, E3 } etype;
etype func( void );
etype evar;
etype func(void)
{
        evar = E1;                      // 10.1 のメッセージが出力される
        return E1;                      // 10.1 のメッセージが出力される
}
```

【回避策】
列挙子を列挙型でキャストしてください。

```
typedef enum E { E1 = 0, E2, E3 } etype;
etype func( void );
etype evar;
etype func(void)
{
        evar = (etype)E1;               // 列挙子を列挙型でキャストする
        return (etype)E1;               // 列挙子を列挙型でキャストする
}
```

11.1.7 デバイス・ファイルを必要とする拡張言語仕様

以下の機能を使用する場合は必ずデバイス・ファイルを指定してください。

- #pragma callt，または __callt
- #pragma saddr，または __saddr
- __near 属性を持つ const 変数

デバイス・ファイルを指定しない場合，不正なコードが生成される可能性があります。

11.1.8 ビット操作命令の出力制御 【V1.04 以降】

組み込み関数を使用せずにビット操作命令を出力したい場合は，次の条件を全て満たしてください。

(a) 定数値を代入する。

(b) 代入先を near 領域の char/unsigned char/signed char/_Bool 型で 1 ビット幅のビットフィールドにする。

(c) 代入先を volatile 修飾する。

volatile 修飾した変数は，上記の条件を満たさない限り，値の代入や読み出しにおいてビット操作命令を出力しません。volatile 修飾していない変数は，コンパイラの最適化に従ってビット操作命令を出力します。

例

```
volatile struct {
  unsigned char bit0:1;
  unsigned int  bit1:1;
} data;

void func(void) {
  data.bit0 = 1; /* ビット操作命令を出力する */
  data.bit1 = 1; /* ビット操作命令を出力しない */
}
```

11.2 ライブラリ・スタートアップの注意事項

この節では，ライブラリ・スタートアップの注意事項について説明します。

11.2.1 プロセッサ・モード・コントロール・レジスタ（PMC）の設定

PMC の設定は，初期設定で 1 度だけ行ってください。初期設定以外での PMC の書き替えは禁止です。

11.2.2 リンカが値を決定するラベル

スタートアップで使用している以下のラベルを定義すると，不正動作となります。これらのラベルは最適化リンカが値を決定するため，定義は不要です。
__STACK_ADDR_START，__STACK_ADDR_END，__RAM_ADDR_START，__RAM_ADDR_END

11.2.3 スタートアップのアセンブル時に必要なオプション

コンパイラドライバを通さずにスタートアップを直接アセンブルする場合は，以下のオプションを指定してください。
-define=__RENESAS_VERSION__=0x01000000 ; CC-RL V1.00 を使用する場合
-define=__RENESAS_VERSION__=0x01010000 ; CC-RL V1.01 を使用する場合

11.2.4 標準ライブラリ関数名の使用制限

標準ライブラリに含まれる関数のうち，以下の関数名はユーザ変数名，またはユーザ関数名としてプログラム中に記述してはいけません。
これらはランタイム・ライブラリに含まれるためです。

- memcpy，memset

11.2.5 標準ライブラリ関数の誤差

- printf, sprintf, vprintf, vsprintf, scanf, sscanf, atof, atoff, strtod, strtof で，浮動小数点数の桁数が多いと誤差が拡大します。
- pow で，べき乗の絶対値が大きいと誤差が拡大します。

11.2.6 bsearch，qsort の K&R 形式の比較関数定義

bsearch，qsort の比較関数定義を K&R 形式で記述した場合，bsearch，qsort の比較関数呼び出し側は実引数を near ポインタとして渡しますが，比較関数定義側は仮引数を far ポインタとして受け取るアセンブラコードを生成するため，プログラムは正しく動作しません。

例

```
#include <stdlib.h>
int table[5] = {0, 1, 2, 3, 4}, key = 3, *ret;
int compare();

void func(void) {
        ret = bsearch(&key, table, 5, sizeof(int), compare);
}

int compare(i, j)
int *i, *j;
{
        if (*i > *j) {
                return 1;
        }
        else if (*i < *j) {
                return -1;
        }
        return 0;
}
```

【回避策】
bsearch，qsort の比較関数の定義と宣言を関数原型形式で記述してください。

```
#include <stdlib.h>
int table[5] = {0, 1, 2, 3, 4}, key = 3, *ret;
int compare(int *, int *);      // 関数原型

void func(void) {
        ret = bsearch(&key, table, 5, sizeof(int), compare);
}

int compare(int *i, int *j)     // 関数原型
{
        if (*i > *j) {
                return 1;
        }
        else if (*i < *j) {
                return -1;
        }
        return 0;
}
```

11.2.7 スタートアップのスタック領域初期化 【V1.07 以前】

スタック領域の初期化処理はコメントアウトされています。未初期化 RAM 読み出しによるパリティ・エラー検出機能を有効にする場合は，スタック領域の初期化処理を有効にしてください。

11.2.8 個別オプションでの C99 規格指定時の標準ライブラリ指定

個別オプションにて -lang オプションで C99 規格を指定し，C99 のソースから C99 規格の標準ライブラリを呼ぶ場合は「リンク・オプション」で C99 用の標準ライブラリを使用するよう指定してください。

11.3 アセンブラの注意事項

この節では，アセンブラの注意事項について説明します。

11.3.1 アセンブラドライバ

アセンブラドライバ (asrl) に複数の入力ファイルを指定した場合に異常終了することがあります。

11.3.2 .DB8 疑似命令

.DB8 疑似命令はシンボルの前方参照を禁止していますが，前方参照してもエラーになりません。前方参照にならないように，シンボルは .DB8 疑似命令より先に定義してください。

例

```
        .DB8    SYM1            ; シンボルの前方参照は禁止
SYM1    .EQU    0x12345         ; シンボルは .DB8 疑似命令より後に定義

SYM2    .EQU    0x12345         ; シンボルは .DB8 疑似命令より先に定義
        .DB8    SYM2            ; シンボルの後方参照は記述可能
```

11.3.3 ビットシンボル

- .EQU 疑似命令でビット位置指定子を用いて定義したシンボルの前方参照がエラーにならず，結果も正しくありません。前方参照にならないように，ビットシンボルは参照より先に定義してください。

例

```
OFFSET   CODE                  NO  SOURCE STATEMENT
00000000                        1
00000000 22220007               2          .DB4    BSYM         ; シンボルの前方参照は禁止
00000004                        3  BSYM    .EQU    0x2222.7
```

- ビットシンボル同士の演算がエラーにならず，結果も正しくありません。ビットシンボル同士の演算を記述しないでください。

例

```
OFFSET   CODE                  NO  SOURCE STATEMENT
00000000                        1
00000000                        2
00000000                        3  BSYM1 .EQU    0x2222.5
00000000                        4  BSYM2 .EQU    0x3333.7
00000000                        5
00000000 11110002               6          .DB4    BSYM2 - BSYM1 ; 記述不可
```

11.3.4 .ALIGN 疑似命令

アブソリュートなセクション (開始アドレスが指定されたセクション) で .ALIGN 疑似命令を記述しても無効になります。

例

```
        .CSEG TEXT
        .ORG    0x1
        .ALIGN 2        ; 無効
_LABEL:
```

11.3.5 分離演算子

【V1.01 のみ】
リロケータブル項に分離演算子を適用した場合，本来 () で囲まれたパラメータのみを演算の対象としますが，() の後ろに記載された値 (シンボル含む) まで含めて演算対象となります。対象となる分離演算子は以下のとおりです。

- HIGH,LOW,HIGHW,LOWW,MIRHW,MIRLW,SMRLW

例

```
_L:
        mov A, !HIGH(_L) + 0x44          ; !HIGH(_L + 0x44) と解釈
```

【V1.02 以降】
リロケータブル項にネストした分離演算子を適用し，分離演算項と定数との加減算を記述した場合，本来 () で囲まれたパラメータを演算の対象としますが，() 内の定数を () の後ろに記載した動作となります。対象となる分離演算子は以下のとおりです。

- HIGH,LOW,HIGHW,LOWW,MIRHW,MIRLW,SMRLW

例

```
_L:
        mov A,!HIGH(LOWW(_L+0x44)+0x55) ; !HIGH(LOWW(_L+0x44))+0x55 と解釈
```

11.3.6 アセンブリ・ソース・ファイルで有効な定義済みマクロ

コンパイルドライバからアセンブラを起動する場合，アセンブリ・ソース・ファイルで定義済みマクロ __RENESAS_VERSION__ が有効になります。
このマクロは "-U __RENESAS_VERSION__" を指定してもアセンブリ・ソース・ファイルで有効になります。

このマクロをアセンブリ・ソース・ファイルで無効にしたい場合は以下のいずれかを行ってください。

- "-asmopt=-undefine=__RENESAS_VERSION__" を指定する。
- アセンブラを直接呼び出す。

11.3.7 指定順序に依存するオプション

オプションの指定順序により -no_warning が効かない場合があります。

例　以下の順で指定すると警告が出ます。
-define=xxxx -undefine=xxxx -no_warning=50649

以下のいずれかの方法で回避してください。

- コンパイルドライバからアセンブラを起動する場合
  - コンパイルオプション "-no_warning_num=50649" を指定する。
  - 他のオプションより前に "-asmopt=-no_warning=50649" を指定する。
- アセンブラを直接起動する場合
  - 他のオプションより前に "-no_warning=50649" を指定します。

    例　以下の順で指定すると警告は出ません。
    -no_warning=50649 -define=xxxx -undefine=xxxx

11.4 リンカの注意事項

この節では，リンカの注意事項について説明します。

11.4.1 -strip オプション

-strip オプションで複数の入力ファイルを指定できません。1 ファイルずつ指定してください。

11.4.2 -memory オプション

以下の条件を全て満たすと，異常終了する場合があります。

- リンク時にオプション -memory=low を指定する。
- リンク時にオプション -nooptimize を指定しない。

【回避策】
オプション -memory=low を指定しないでください。

11.4.3 変数 / 関数情報ファイルの上書き

-vfinfo オプションの指定により出力された変数 / 関数情報ファイルを編集した後，2 回目のリンクで同じファイル名で -vfinfo オプションを指定すると，変数 / 関数情報ファイルを上書きします。
編集した変数 / 関数情報ファイルを別名で保存するか，編集後は -vfinfo オプションを指定しないか，-vfinfo オプションの出力ファイル名を変更してください。

11.4.4 セクション配置

セクション自動配置 (-auto_section_layout) オプション未指定時は，.SECTION/.CSEG/.DSEG 疑似命令で指定した再配置属性に対応した領域 (アドレス範囲) にセクションが自動で配置されません。また，64KB-1 境界をまたがらないような配置も行いません。
リンカの -start オプションで各セクションの配置先として適切なアドレスを指定してください。

11.4.5 コンパイル・エラーを引き起こす可能性のある変数 / 関数情報ファイル

__inline 指定されたグローバルな関数を含むプログラムのリンク時に -vfinfo オプションを指定する場合，出力された変数 / 関数情報ファイルをインクルードしてコンパイルする時にエラーになる場合があります。
このエラーは変数 / 関数情報ファイルが上記関数に対する "#prgma __callt" を含む時に起こります。

このエラーを回避するためには以下のいずれかを行ってください。

- 変数 / 関数情報ファイルを編集して当該関数に対する指定をコメント化，又は，削除する。
- __inline キーワードを削除する。
- -vfinfo オプションを指定しない。

11.4.6 saddr アクセス範囲外のアドレスに関するエラー出力

-dev オプションを指定している場合は saddr アクセス範囲外のアドレスに対してエラーを出力しますが，-dev オプションを指定していない場合はエラーを出力しません。

例 >ccrl -cpu=S3 file.asm -lnkopt=-start=.text/100,.bss/f9f34

```
        mov     a,!sym
        mov     a,sym           ; -dev オプション指定時はエラー出力

        .section .bss, bss
sym:    .DS     (1)
```

11.4.7 コンパイラ・パッケージのバージョン

最適化リンカを使用する際は，入力するすべてのオブジェクト・ファイル，リロケータブル・ファイル，ライブラリ・ファイルを生成したコンパイラ・パッケージと同じか，より新しいコンパイラ・パッケージに付属しているものを使用してください。

標準ライブラリおよびランタイム・ライブラリを使用する際は，使用する最適化リンカと同じコンパイラ・パッケージに付属しているものを使用してください。

A. クイック・ガイド

この章では，CC-RL 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-RL では変数や定数に far 属性，near 属性，または saddr 属性のいずれかの属性を指定した上で，各々を各属性と同じ名前を持つ領域，すなわち，far 領域，near 領域，または saddr 領域に配置します。
変数や定数をアクセスする命令のサイズはこの順番で小さくなるため，変数や定数を near 属性（または saddr 属性）にして near 領域（または saddr 領域）に配置するとコードサイズが小さくなります。
一方，領域のサイズもこの順番で小さくなるため，変数や定数の合計サイズが大きい場合には各変数をどの領域に配置するかの優先順位を付ける必要があります。

以下では，各属性の指定方法，各領域とその配置方法及びについて詳しく説明します。

(1) 変数や定数に属性に指定する方法
変数や定数にはオプションやキーワードを使って，属性を指定します。
指定方法は以下の 4 つで，優先度はこの順番で高くなります。
詳細は各オプションやキーワード，および，「2.6.6 near，far との関係」を参照してください。

(a) -cpu オプション
ただし，変数や定数のデフォルト属性は常に near 属性

(b) -memory_model オプション
ただし，変数や定数のデフォルト属性は常に near 属性

(c) -far_rom オプション
ROM データはデフォルトで far 属性になる

(d) __near/__far/_saddr キーワード
変数は各キーワードで指定された属性になる

例 1. -far_rom オプション非指定時

```
long                x;      //near 属性
long __near         y;      //near 属性
long __far          z;      //far 属性
const int           c;      //near 属性
const int __near    d;      //near 属性
const int __far     e;      //far 属性
char __saddr        s;      //saddr 属性
```

例 2. -far_rom オプション指定時

```
long                x;      //near 属性
long __near         y;      //near 属性
long __far          z;      //far 属性
const int           c;      //far 属性
const int __near    d;      //near 属性
const int __far     e;      //far 属性
char __saddr        s;      //saddr 属性
```

(2) 変数や定数を各領域に配置する優先度
一般的には，変数の静的参照回数が多い変数を，命令サイズが小さな領域に割り付けるとプログラム全体のコードサイズが小さくなります。
しかし，命令サイズが小さな領域はその領域サイズも小さいため，効率的に使う必要があります。
そこで，領域 1 バイトあたりの削減効率を最大化することを考えると，変数の静的参照回数を変数サイズで割った値が大きい変数ほど命令サイズが小さな領域に割り付けると良いとわかります。
そこで，優先度は以下と考えられます。

```
優先度 = 変数の静的参照回数 / 変数サイズ
```

ここで，変数の静的参照回数は，オブジェクト・コード上の参照回数を指し，C ソース上の参照回数とは異なります。
RL78 では変数参照に使える命令は 1 バイトや 2 バイトのアクセス幅のみのため，例えば 4 バイト変数を参照するには 2 つの命令が必要になります。
したがって，C ソース上で 4 バイト変数を 1 回参照すると，オブジェクト・コード上では 2 回参照することになります。
オブジェクト・コード上の参照回数はリンカに -show=reference オプションを指定することで表示することができます。

(3) 領域
各領域はアドレス範囲によって 3 つに分けられます。
far 領域は，saddr 領域や near 領域を含む全領域となります。

(a) saddr 領域
アドレスが 0xFFE20 から 0xFFF1F までの領域を saddr 領域と呼びます。
このアドレス範囲に配置した saddr 属性変数はサイズが最も小さい命令でアクセスします。
saddr 領域は，特殊機能レジスタ (SFR) の一部や汎用レジスタも含みます。
SFR や汎用レジスタには変数を配置できません。

(b) near 領域
アドレスが 0xF0000 から 0xFFE1F までの領域を near 領域と呼びます。
この領域には内部 RAM，ミラー領域，データ・フラッシュ・メモリを含みます。
このアドレス範囲に配置した near 属性変数は saddr 領域に次いでサイズが小さい命令でアクセスします。
near 領域は，拡張特殊機能レジスタ (2nd SFR) も含みます。
2nd SFR には変数を配置できません。

(c) far 領域
saddr 領域や near 領域を含む全領域を far 領域と呼びます。
この領域には内部 RAM 以外の RAM やミラー領域以外のコード・フラッシュ・メモリ内の定数領域も含みます。
このアドレス範囲に配置した far 属性変数はサイズが最も大きい命令でアクセスします。

(4) 変数や定数を各領域に配置する方法
各変数や定数を含むセクションの配置アドレスをリンク・オプションを使って指定することによって，変数や定数を各領域に配置します。

例 1. .saddr 領域に配置するデフォルトのセクションは .sbss と .sdata です。
.sbss は初期値無の saddr 属性変数を配置するセクション，.sdata は初期値を持つ saddr 属性変数を配置するセクションです。
ROM イメージを作成する時，.sdata を任意のアドレスに配置し，.sdata 上のデータを RAM へ転送する場合の転送先セクションの名前を適当に決めます（以下では .sdataR としています）。

```
-rom=.sdata=.sdataR
-start=.sdata/5000
-start=.sbss,.sdataR/ffe20
```

例 2. near 領域に配置するデフォルトのセクションは .bss，.data，および .const です。
.bss は初期値無の near 属性変数を配置するセクション，.data は初期値を持つ near 属性変数を配置するセクションです。
ROM イメージを作成する時，.data を任意のアドレスに配置し，.data 上のデータを RAM へ転送する場合の転送先セクションの名前を適当に決めます（以下では .dataR としています）。
.const は near 属性定数を配置するセクションです。
near 属性定数を配置する領域のアドレスはデバイスごとに決まっているので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。

```
-rom=.data=.dataR
-start=.const/2000
-start=.data/4000
-start=.bss,.dataR/fe000
```

例 3. far 領域に配置するデフォルトのセクションは .bssf，.dataf，および .constf です。
RAM として内部 RAM のみを使うのであれば .bssf や .dataf は通常は使いません。
そこで以下では .constf のみ説明します。
.constf は far 属性定数を配置するセクションです。
.constf はコード・フラッシュ・メモリ上に配置します。

```
-start=.constf/6000
```

A.1.2 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，アクセス幅，およびアクセス回数も変更されません。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。

```
int a;
int b;
void func(void){
    if(a <= 0){
        b++;
    } else {
        b+=2;
    }
    b++;
}
```

```
_func:
        movw    ax, !LOWW(_a)
        xor     a, #0x80
        cmpw    ax, #0x8001
        movw    ax, !LOWW(_b)
        bnc     $.BB@LABEL@1_3
.BB@LABEL@1_1:          ; bb1
        incw    ax
.BB@LABEL@1_2:          ; bb9
        incw    ax
        movw    !LOWW(_b), ax
        ret
.BB@LABEL@1_3:          ; bb3
        incw    ax
        br      $.BB@LABEL@1_1
```

例 2. volatile 指定した場合のソースと出力コードの例
“ 変数 a”，および “ 変数 b” 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。
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:
        movw    ax, !LOWW(_a)
        xor     a, #0x80
        cmpw    ax, #0x8001
        bnc     $.BB@LABEL@1_2
.BB@LABEL@1_1:          ; bb1
        onew    ax
        br      $.BB@LABEL@1_3
.BB@LABEL@1_2:          ; bb3
        movw    ax, #0x0002
.BB@LABEL@1_3:          ; bb3
        addw    ax, !LOWW(_b)
.BB@LABEL@1_4:          ; bb9
        movw    !LOWW(_b), ax
        incw    !LOWW(_b)
        ret
```

A.1.3 const 定数ポインタを定義する

ポインタについては，“const” の指定位置により，異なる解釈がされます。

- 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/.constf）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），および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/.constf）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 指令を使用します。

```
#pragma section text  " セクション名 "
```

#pragma section 指令で，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，セクションが near の場合は，“_n” が追加されたもの，セクションが far の場合は，“_f” が追加されたものになります。
セクション種別 "text" を省略した場合や，セクション種別指定なしとありが混在した場合 については「コンパイラ出力セクション名の変更（#pragma section）」を参照してください。

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=Mytext_n/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。
-start オプションは，リンク・オプションです。詳細は「-STARt オプション」を参照してください。

A.2.2 アセンブラ命令の埋め込み

CC-RL 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列とみなして，呼び出し箇所にインライン展開します。

- #pragma 指令

```
#pragma inline_asm func
static int func(int a, int b) {
        /* アセンブラ命令 */
}
```

詳細は「アセンブラ命令の記述（#pragma inline_asm）」を参照してください。

A.2.3 プログラムを RAM 上で実行する方法

ROM 内に配置しているプログラムを RAM に転送して，RAM 上で実行することができます。
転送されるプログラムの属性は far 属性にします。
far 属性を持つデフォルトのテキスト・セクションは .textf になります。
.textf 全体でなく，その一部を RAM 上で実行したい場合，まず #pragma section でそのセクション名を変更し，-rom オプションでその変更後のセクション名を指定します。
次に，変更後のセクションを ROM から RAM に転送することによってそのセクションを RAM 上で実行することができます。

例　　割り込みがある場合に f1 と f2 を RAM へ転送し，RAM 上で実行します。

- ファイル：ram.c

```
#include “iodefine.h”
#pragma section text    ram_text

__far void f1(char) {...}
__far int f2(int) {...;f1(x);...}

#pragma section
#pragma interrupt       inthandler (vect=INTP0)

void inthandler(void){
    /*ram_text_f セクションから ram_text_fR セクションへプログラムを転送する */
        unsigned char __far *dst, *src;
        src = __sectop("ram_text_f");
        dst = __sectop("ram_text_fR");
        while (src < __secend("ram_text_f")) {
                *dst++ = *src++;
        }
    /* 転送した RAM 上のプログラムを呼び出す */
        f2(1);
}
```

- リンク・オプション

```
-rom=ram_text_f=ram_text_fR
-start=ram_text_f/3000
-start=ram_text_fR/ff000
```

A.3 変数（アセンブリ言語）

この節では，変数（アセンブリ言語）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  シンボル名
```

例　　初期値なし変数定義

```
        .DSEG   sbss
        .PUBLIC _val0           ;_val0 を他ファイルから参照可能にします
        .PUBLIC _val1           ;_val1 を他ファイルから参照可能にします
        .PUBLIC _val2           ;_val2 を他ファイルから参照可能にします
        .ALIGN  2               ;_val0  を 2 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 疑似命令 /.DB4 疑似命令を使用します。
初期値あり変数については，8.4 ROM イメージの作成 も参照してください。

- 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  シンボル名
```

例　　初期値あり変数定義

```
        .DSEG   sdata
        .PUBLIC _val0           ;_val0 を他ファイルから参照可能にします
        .PUBLIC _val1           ;_val1 を他ファイルから参照可能にします
        .PUBLIC _val2           ;_val2 を他ファイルから参照可能にします
        .ALIGN  2               ;_val0 を 2 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              ;_val1 は 2 バイト分領域を確保し，10 を格納します
_val2:
        .DB     1               ;_val2 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/.constf セクション中で，.DB 疑似命令 /.DB2 疑似命令 /.DB4 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              ;_p を他ファイルから参照可能にします
        .ALIGN  2               ;_p を 2 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	2014.08.01	-	初版発行
1.01	2015.02.01	391, 他	アセンブラ再配置属性 SBSS_BIT，BSS_BIT，BIT_AT を追加しました。
		399, 400, 他	アセンブラ疑似命令 .BSEG を追加しました。
		413, 他	アセンブラ疑似命令 .DBIT を追加しました。
		130	アセンブラ移行支援機能において，以下の記述をサポートしました。 BSEG 再配置属性なし，BSEG UNIT，BSEG AT，DBIT
		131	移行支援を強化するため，アセンブラ移行支援機能の以下の疑似命令で複数オペランドの記述を可能にしました。 DB，DW，DG
		131	移行支援を強化するため，”$INCLUDE(” 直後に空白（以下，△とする）が書ける記述に対応しました。 “$INCLUDE(△ a.inc)” は ”$INCLUDE(a.inc)” と同じ意味となります。
		268	アセンブラにおいて以下のマクロを追加しました。 __RENESAS_VERSION__ ：バージョンが V.XX.YY.ZZ の場合，0xXXYYZZ00 とします。
		463	標準ライブラリ，およびランタイムライブラリコード用セクション .SLIB，.RLIB を追加しました。
		536, 538, 542, 544	以下のライブラリ関数，およびマクロを追加しました。 printf_tiny，sprintf_tiny，__PRINTF_TINY__
		508, 509	ライブラリ関数 ldexp，および ldexpf における NaN に対する処理を以下のように変更しました。 ・入力引数が NaN である場合，NaN を返し，グローバル変数 errno にマクロ EDOM を設定します。
		268, 269	以下のマクロを追加しました。 __CNV_CA78K0R__，__CNV_NC30__，__CNV_IAR__，__BASE_FILE__ 以下のマクロを -ansi オプション指定時のみ有効となるように修正しました。 __STDC_VERSION__
		268	以下のマクロ定義の記述誤りを修正しました。 値は設定されません。→ 10 進定数 1 __RENESAS__，__RL78__，__RL78_S1__，__RL78_S2__，__RL78_S3__，__RL78_SMALL__，__RL78_MEDIUM__，__CCRL__，__CCRL，__DBL4，__DBL8，__SCHAR，__UCHAR，__SBIT，__UBIT，__FAR_ROM__
		274	可変個引数マクロを許可しました。
		288, 289, 他	キーワード __callt を追加しました。
		322, 323	#pragma callt を追加しました。
		321	#pragma saddr を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.01	2015.02.01	324, 301, 310, 319	C 言語構文中，および以下の #pragma 指令で 2 進定数を記述可能としました。 #pragma interrupt（vect のパラメータ） #pragma rtos_interrupt（vect のパラメータ） #pragma address（アドレス）
		326	組み込み関数 __get_psw，__set_psw を追加しました。
		310, 他	#pragma rtos_interrupt の vect 指定を省略可能な記述方法に変更し，vect 未指定時の仕様を追加しました。
		301, 305, 310	#pragma interrupt，および #pragma rtos_interrupt の仕様を変更しました。 ・vect 指定時，割り込みハンドラは強制的に __near 属性にします。 ・vect 未指定時，割り込みハンドラの __near/__far 属性は変更しません。 #pragma interrupt_brk の仕様を変更しました。 ・割り込みハンドラは強制的に __near 属性にします。
		280	ビットフィールドの配置方法に関する説明の記述誤りを修正しました。
		307	#pragma section において，セクション種別を指定する場合のみ，”.”（ドット）を変更セクション名の先頭に使用できるように仕様を変更しました。
		305	#pragma interrupt_brk の多重割り込み許可指定に関する説明から DI 命令の生成に関する記述を削除しました。
		76, 261	C 言語における多バイト文字，および -character_set オプションの指定形式における gb2312 を gbk に変更しました。
		24, 70	コンパイル・オプション -pack を追加しました。
		24, 91 ～ 101	コンパイル・オプション -convert_cc を追加しました。
		24, 78, 80, 81	以下のコンパイル・オプションを Professional 版で使用可能なオプションとしました。 -misra2004，-ignore_files_misra，-check_language_extension
		786	標準ライブラリ関数名の使用制限の記述誤りを修正しました。
		173	リンク・オプション -VFINFO を追加しました。
		191	リンク・オプション -AUTO_SECTION_LAYOUT を追加しました。
		192	リンク・オプション -DEBUG_MONITOR を追加しました。
		193	リンク・オプション -RRM を追加しました。
		194	リンク・オプション -SELF を追加しました。
		195	リンク・オプション -SELFW を追加しました。
		196	リンク・オプション -OCDTR を追加しました。
		197	リンク・オプション -OCDTRW を追加しました。
		198	リンク・オプション -OCDHPI を追加しました。
		199	リンク・オプション -OCDHPIW を追加しました。
		202	リンク・オプション -CHECK_DEVICE を追加しました。
		203	リンク・オプション -CHECK_64K_ONLY を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.01	2015.02.01	204	リンク・オプション -NO_CHECK_SECTION_LAYOUT を追加しました。
		176, 177	リンク・オプション -show=struct を指定した場合は，構造体および共用体メンバの情報をリンク・マップ・ファイルに出力するようにしました。
		461	リンク・オプション指定により生成する，以下のシンボルを追加しました。 __STACK_ADDR_START, __STACK_ADDR_END, __RAM_ADDR_START, __RAM_ADDR_END, .monitor1, .monitor2
		758	奇数番地を 2 バイト以上の型で間接参照する場合，可能な範囲で警告を出力するようにしました。
		611	スタートアップにおけるスタック領域の設定仕様を変更しました。
		781	20 件の注意事項を追加しました。
		317, 318	#pragma inline_asm に例を追加しました。
		430, 431	.EXITM, .EXITMA 疑似命令の詳細説明を変更しました。
		408, 409, 他	アセンブラ言語仕様に例を追加しました。
1.02	2015.09.14	23, 34, 他	コンパイル・オプション -g_line を追加しました。
		24, 65, 他	コンパイル・オプション -stack_protector/-stack_protector_all を追加しました。
		24, 82, 他	コンパイル・オプション -misra2012 を追加しました。
		24	-ignore_files_misra と -check_language_extension の説明に MISRA-C:2012 ルールを追記しました。
		56	最適化項目に same_code を追加しました。
		84	［注意］に MISRA-C:2012 ルールに関する記述を追加しました。
		85	-misra2012 を追加しました。
		87	MISRA-C:2012 ルールを追記しました。
		88	MISRA-C:2012 ルールを追記しました。
		150, 186, 他	リンク・オプション -SYmbol_forbid を追加しました。
		156	［制限］を追加しました。
		174	-CRc の［指定形式］および［詳細説明］を変更しました。
		187	-Optimize の［指定形式］,［詳細説明］および［備考］を変更しました。
		237	-misra2004 の記述場所を変更しました。
		238	機能が矛盾するオプションの組み合わせに -misra2004 と -misra2012 を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.02	2015.09.14	280	整数定数の型に unsigned long long int を追加しました。
		293, 294, 他	#pragma stack_protector，#pragma no_stack_protector を追加しました。
		426	オペランドとして記述できないシンボルの内容を変更しました。
		427	オペランドに関する内容を変更しました。
		428	オペランドに関する内容を変更しました。
		432	マクロ疑似命令から .EXITM，.EXITMA を削除しました。
		436	仮パラメータの最大数とマクロ呼び出しについての内容を変更しました。
		437	［詳細説明］を変更しました。
		438	文の並び（ブロック）がない場合についての記述を削除しました。
		439	［詳細説明］を変更しました。
		440	［詳細説明］を変更しました。
		441	［詳細説明］を変更しました。
		448	［詳細説明］を変更しました。
		466	アセンブラ生成シンボルに @$IMM_ 定数値を追加しました。
		569, 580, 他	以下のライブラリ関数を追加しました。 calloc，free，malloc，realloc
		631	スタートアップ・ルーチンの例を変更しました。
		649	E0520014 に [対処方法] を追加しました。
		650	E0520020 に [対処方法] を追加しました。
		701	E0523005 に [対処方法] を追加しました。
		722	E0562200 に [対処方法] を追加しました。
		722	E0562201 に [対処方法] を追加しました。
		723	E0562320 に [対処方法] を追加しました。
		733	F0523029 の［メッセージ］と［説明］を変更しました。
		733	F0523030 の［説明］を変更しました。
		746	F0563113 に［対処方法］を追加しました。
		753	M0523086 を追加しました。
		757	W0511179 に［対処方法］を追加しました。
		800	K&R 形式に関する注意事項を変更しました。
		805	分離演算子に関する注意事項を変更しました。
1.03	2016.07.01	55, 56	最適化項目に pipeline を追加しました。
		73	［詳細説明］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.03	2016.07.01	85	次の MISRA-C:2012 ルールを追加しました。 2.6 2.7 9.2 9.3 12.1 12.3 12.4 14.4 15.1 15.2 15.3 15.4 15.5 15.6 15.7 16.1 16.2 16.3 16.4 16.5 16.6 16.7 17.1 17.7 18.4 18.5 19.2 20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12 20.13 20.14
		174	［詳細説明］を変更しました。
		210	［備考］を変更しました。
		237	-Opipeline を追加しました。
		333	［機能］と［制限］を変更しました
		336	組み込み関数 __set1，__clr1，__not1 を追加しました。
		472, 473	【V1.03 以降】の内容を追加しました。
		581	［詳細説明］を変更しました。
		582	［詳細説明］を変更しました。
		583	［詳細説明］を変更しました。
		584	［詳細説明］を変更しました。
		644, 他	不要なメッセージを削除しました。
		742	W0520171 を追加しました。
1.04	2016.12.01	13	「ライセンスについて」の説明を変更しました。
		13	「standard 版と professional 版について」を追加しました。
		13	「無償評価版について」を追加しました。
		20, 21	説明を変更しました。
		54	［詳細説明］を変更しました。
		56- 58	［使用例］を変更しました。
		74	［詳細説明］を変更しました。
		86, 87	次の MISRA-C:2012 ルールを追加しました。 2.2 3.2 5.1 5.6 5.7 5.8 5.9 8.3 8.9 9.1 12.2 21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 21.10
		182	［備考］を変更しました。
		184	［詳細説明］を変更しました。
		268, 269	説明を変更しました。
		270, 他	処理系依存の次の項目を変更しました。 4.1.3 (1), (4), (6), (7), (9), (12), (14), (16), (36), (37), (38), (39)
		282	説明を変更しました。
		328	［制限］を変更しました。
		329	［使用例］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.04	2016.12.01	331	［制限］を変更しました。
		335	［機能］を変更しました。
		434	［注意事項］を変更しました。
		435	［注意事項］を変更しました。
		436	［詳細説明］を変更しました。
		448	「機械語命令セット」を追加しました。
		584	［詳細説明］を変更しました。
		585	［詳細説明］を変更しました。
		586	［詳細説明］を変更しました。
		587	［詳細説明］を変更しました。
		597	［注意事項］を追加しました。
		614	［注意事項］を追加しました。
		622-629	説明を変更しました。
		631, 632	説明を変更しました。
		650	E0520020 を変更しました。
		652	E0520065 を変更しました。
		701	E0551406 を追加しました。
		724	不要なメッセージを削除しました。
		732	F0563006 を追加しました。
		732	F0563020 を変更しました。
		733	F0563115 を追加しました。
		741	M0560700 を追加しました。
		755	不要なメッセージを削除しました。
		764	W0561014 を追加しました。
		783, 784	「ビット操作命令の出力制御」を追加しました。
		786	「スタートアップのスタック領域初期化」を追加しました。
		794	説明を変更しました。
1.05	2017.06.01	13	「standard 版と professional 版について」の説明を変更しました。
		24, 66, 77	コンパイル・オプション -insert_nop_with_label を追加しました。
		61	［詳細説明］を変更しました。
		70	［詳細説明］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.05	2017.06.01	87	［指定形式］を変更しました。
		87, 88	次の MISRA-C:2012 ルールを追加しました。 12.5 13.2 13.5 17.5 17.8 21.13 21.15 21.16
		147	［詳細説明］を変更しました。
		148	［詳細説明］を変更しました。
		151, 162, 169	リンク・オプション -END_RECORD を追加しました。
		158	［詳細説明］を変更しました。
		164	表 2.9 を変更しました。
		177	［使用例］を変更しました。
		185	［指定形式］,［詳細説明］を変更しました。
		186	［備考］を変更しました。
		189	［詳細説明］を変更しました。
		190	［備考］を変更しました。
		205	［詳細説明］を変更しました。
		215	［備考］を変更しました。
		222	［備考］を追加しました。
		248, 250, 251	「リンク・マップ情報」の説明を変更しました。
		271, 272	「変数 / 関数情報ファイルの出力」の説明を変更しました。
		272	「変数 / 関数情報ファイルの利用」の説明を変更しました。
		281, 299, 300	「#pragma 指令」の説明を変更しました。
		300, 339, 340	「near/far 関数（#pragma near/#pragma far）」を追加しました。
		300, 341	「構造体のパッキング（#pragma pack/#pragma unpack）」を追加しました。
		307	「キーワードおよび #pragma との関係」を変更しました。
		332	［制限］を変更しました。
		334, 335	［制限］を変更しました。
		337	［機能］,［制限］を変更しました。
		342	［方法］を変更しました。
		427	［指定形式］,［詳細説明］を変更しました。
		428	［指定形式］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.05	2017.06.01	429	［指定形式］を変更しました。
		430	［指定形式］を変更しました。
		629-632	次の割り込み禁止時間を変更しました。 acos，acosf，asin，asinf，atan，atanf，atan2，atan2f，tan，tanf， cosh，coshf，sinh，sinhf，tanh，tanhf，exp，expf，pow，powf， sqrt，sqrtf，scanf，sscanf，atof，_COM_atof_f，atoff，_COM_atoff_f， strtod，_COM_strtod_ff，strtof，_COM_strtof_ff
		634	_COM_fdiv の割り込み禁止時間を変更しました。
		653, 654, 728, 739, 751, 755	次のメッセージを追加しました。 C0511200，C0519996，C0519997，C0550802，C0550804， C0550805，C0550806，C0550808，C0551800，C0564001， F0563103，W0511184，W0511185，W0523120，W0561015，W0561016
		657, 720, 727, 755	次のメッセージを変更しました。 E0511200，F0523073，F0563006，W0561004，W0561017
		755	不要なメッセージを削除しました。
		780	「saddr アクセス範囲外のアドレスに関するエラー出力」を追加しました。
1.06	2017.12.01	11,13	C99 規格に対応しました。
		11	「最大値」を削除しました。
		23, 40, 41, 254, 297-303, 308, 329	コンパイル・オプション -lang を追加しました。
		24, 62, 64, 254	コンパイル・オプション -change_message を追加しました。
		24, 69, 80-82, 254	コンパイル・オプション -control_flow_integrity を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.06	2017.12.01	24, 84, 85, 254, 286, 288, 297-303, 308, 311, 323, 326, 328, 358	コンパイル・オプション -strict_std を追加しました。
		25, 105, 117, 254	コンパイル・オプション -unaligned_pointer_for_ca78k0r を追加しました。
		54	コンパイル・オプション -far_rom の［詳細説明］を変更しました。
		56	コンパイル・オプション -O の［詳細説明］を変更しました。
		63	コンパイル・オプション -no_warning_num の［指定形式］と［詳細説明］を変更しました。
		92	コンパイル・オプション -misra2004 の［注意］を変更しました。
		93, 94	次の MISRA-C:2012 ルールを追加しました。 8.14 9.4 9.5 13.1 17.6 18.7 21.11 21.12
		106, 107	コンパイル・オプション -convert_cc の［詳細説明］を変更しました。
		114	コンパイル・オプション -convert_cc=iar 指定時の #pragma pack の動作を変更しました。
		154	アセンブル・オプション -warning の［指定形式］を変更しました。
		155	アセンブル・オプション -no_warning の［指定形式］を変更しました。
		159, 170, 172, 173, 180, 185	リンク・オプション -FIX_RECORD_LENGTH_AND_ALIGN を追加しました。
		159, 170, 172, 196	リンク・オプション -CFI を追加しました。
		159, 170, 172, 197	リンク・オプション -CFI_ADD_Func を追加しました。
		159, 170, 172, 198	リンク・オプション -CFI_IGNORE_Module を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.06	2017.12.01	166	リンク・オプション -Binary の［制限］を変更しました。
		173, 184	リンク・オプション -BYte_count の使用可能条件を変更しました。
		201, 202, 260, 266	リンク・オプション -SHow の引数に CFI を追加しました。
		207	リンク・オプション -Absolute_forbid に［注意］を追加しました。
		294	「処理系依存」の「あらかじめ定義されたマクロ名」を変更しました。
		302	「ANSI 規格に厳密な処理オプション」を削除しました。
		312- 313	「マクロ」を追加しました。
		331- 332	「ハードウェア割り込みハンドラ（#pragma interrupt）」の［方法］を変更しました。
		347	「アセンブラ命令の記述（#pragma inline_asm）」の［制限］にアセンブラの制御命令および疑似命令に関する記述を追加しました。
		499	以下のヘッダ・ファイルを追加しました。 inttypes.h，iso646.h，stdbool.h
		644, 651	__control_flow_integrity 関数を追加しました。
		714, 735, 757	次のメッセージを追加しました。 E0523087，F0563602，W0561331
		685, 723- 726, 732- 733, 735 738 741, 755- 757	次のメッセージを変更しました。 E0520411，E0562311，E0562340，E0562350，E0562351，E0562352， E0562353，E0562354，E0562355，E0562360，E0562361，E0562362， E0562363，E0562364，E0562365，E0562417，F0563004，F0563102， F0563123，F0563124，F0563431，M0560005，W0520062，W0561130， W0561184，W0561325，W0561531
		669	次の内部エラーメッセージを削除しました。 C0560901，C0560903，C0560904，C0560905，C0560906，C0560907， C0592xxx，C0592100，C0592200

Rev.	発行日	改定内容	
		ページ	ポイント
1.06	2017.12.01	670, 714, 720-726	次のエラーメッセージを削除しました。 E0511120, E0544003, E0544240, E0544854, E0560601, E0560602, E0560603, E0560604, E0560605, E0560606, E0560607, E0560608, E0560609, E0560610, E0560611, E0560612, E0560613, E0560614, E0560615, E0560616, E0560617, E0560618, E0560619, E0560620, E0560621, E0560622, E0560623, E0560624, E0560637, E0560638, E0560641, E0560660, E0562015, E0562017, E0562021, E0562112, E0562113, E0562143, E0562203, E0562222, E0562223, E0562323, E0562324, E0562331, E0562366, E0562400, E0562402, E0562403, E0562404, E0562405, E0562406, E0562407, E0562408, E0572000, E0572200, E0572500, E0572501, E0572502, E0573005, E0573007, E0573008, E0573009, E0573300, E0573303, E0573310, E0573320, E0573500, E0573505, E0592001, E0592002, E0592003, E0592004, E0592005, E0592006, E0592007, E0592008, E0592010, E0592013, E0592015, E0592016, E0592018, E0592019, E0592020, E0592101, E0592102, E0592201, E0593002, E0593003, E0593004, E0594000, E0594001, E0594002
		730-735	次の致命的エラーメッセージを削除しました。 F0542001, F0542002, F0544302, F0544802, F0560001, F0560002, F0560003, F0560004, F0560005, F0560006, F0560007, F0560008, F0560009, F0560010, F0560011, F0560012, F0560013, F0560101, F0560102, F0560103, F0560104, F0560105, F0560106, F0560107, F0560108, F0560109, F0560110, F0560112, F0560113, F0560114, F0560115, F0560201, F0560202, F0560203, F0560204, F0560208, F0560209, F0560210, F0560213, F0560215, F0560216, F0560217, F0560218, F0560219, F0560220, F0560301, F0560302, F0560303, F0560304, F0560306, F0560307, F0560309, F0560310, F0560311, F0560404, F0560405, F0560407, F0560409, F0560411, F0560414, F0560415, F0560417, F0560419, F0560421, F0560423, F0560424, F0560502, F0560503, F0560627, F0560629, F0560630, F0560631, F0560633, F0560634, F0560635, F0560636, F0560649, F0560650, F0560652, F0560657, F0560658, F0560661, F0560662, F0560701, F0560705, F0560707, F0560708, F0560712, F0561001, F0561002, F0561003, F0561004, F0561005, F0561006, F0561007, F0561008, F0561009, F0561010, F0561011, F0561012, F0561013, F0561014, F0561015, F0561016, F0561019, F0562001, F0562002, F0562003, F0562004, F0562005, F0562006, F0562007, F0562008, F0562009, F0562014, F0562028, F0563120, F0563311, F0563312, F0563313, F0563400, F0563420, F0578200, F0578201, F0578202, F0578203, F0578204, F0578205, F0578206, F0578207, F0578208, F0578209, F0578210, F0578212, F0578213, F0578214, F0578215, F0578216, F0578217, F0578218, F0578219, F0578220, F0578221, F0593113, F0593114, F0595001, F0595002, F0595003, F0595004
		738	次のインフォメーションメッセージを削除しました。 M0560001, M0560002, M0560102, M0560103, M0560300, M0560510, M0560511, M0560512, M0560600, M0592150, M0592151, M0592152, M0592153, M0592154, M0592155, M0592156, M0592157, M0592250, M0592251, M0592252, M0592253, M0592270, M0592280, M0592281, M0592282, M0594201, M0594202, M0594203

Rev.	発行日	改定内容	
		ページ	ポイント
1.06	2017.12.01	752-757	次のワーニングメッセージを削除しました。 W0542101，W0544001，W0544002，W0560111，W0560116，W0560205， W0560206，W0560207，W0560212，W0560214，W0560305，W0560308， W0560312，W0560313，W0560314，W0560315，W0560316，W0560401， W0560402，W0560403，W0560406，W0560408，W0560410，W0560412， W0560413，W0560416，W0560418，W0560420，W0560422，W0560501， W0560625，W0560628，W0560639，W0560640，W0560642，W0560643， W0560644，W0560645，W0560647，W0560651，W0560653，W0560654， W0560655，W0560656，W0560659，W0560702，W0560706，W0560709， W0560710，W0560711，W0561110，W0561180，W0561190，W0561192， W0561194，W0561304，W0561321，W0561327，W0561420，W0561430， W0561500，W0561501，W0561502，W0561510，W0562010，W0562011， W0562012，W0562013，W0562015，W0562016，W0562017，W0562018， W0562019，W0562020，W0562021，W0562022，W0562023，W0562024， W0562025，W0562026，W0562027，W0571600，W0578306，W0578307， W0578308，W0578309，W0578310，W0578311，W0578315，W0578322， W0592009，W0592011，W0592012，W0592017，W0592103，W0592104， W0592105，W0594100，W0594101，W0594102，W0594103，W0594104， W0594105，W0594106，W0594107，W0594110，W0594111，W0594112， W0594113，W0594114，W0594115，W0594116，W0594117，W0594118， W0594119，W0594124，W0594125，W0594126，W0594127，W0594128， W0594129，W0594130，W0594131，W0594132，W0594133，W0594134， W0594135，W0594140，W0594150，W0594151，W0594160，W0594161， W0594162，W0594163，W0594164，W0594165，W0594166
		761	「ビット操作命令の出力制御」を変更しました。
		769	「短い命令長でアクセスできる領域へ配置する」の説明を変更しました。
1.07	2018.06.01	19	「指定形式」の説明を変更しました。
		41	コンパイル・オプション -lang に［備考］を追加しました。
		51	コンパイル・オプション -preprocess の［詳細説明］を変更しました。
		63	コンパイル・オプション -no_warning_num の［指定形式］を変更しました。
		64	コンパイル・オプション -change_message の［指定形式］を変更しました。
		95	コンパイル・オプション -ignore_files_misra の［詳細説明］を変更しました。
		96	コンパイル・オプション -check_language_extension の［詳細説明］を変更しました。
		154	アセンブル・オプション -warning の［指定形式］を変更しました。
		155	アセンブル・オプション -no_warning の［指定形式］を変更しました。
		159, 170, 194	リンク・オプション -SPLIT_VECT を追加しました。
		160, 227, 232	リンク・オプション -CHECK_OUTPUT_ROM_AREA を追加しました。
		163	リンク・オプション -Input の［指定形式］を変更しました。
		166	リンク・オプション -Binary の［指定形式］を変更しました。
		168	リンク・オプション -DEFine の［指定形式］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	171-173	リンク・オプション -FOrm の［指定形式］，［詳細説明］，［備考］を変更しました。
		176	リンク・オプション -RECord の［指定形式］を変更しました。
		177	リンク・オプション -END_RECORD の［指定形式］を変更しました。
		179	リンク・オプション -OUtput の［指定形式］，［詳細説明］，［備考］を変更しました。
		182	リンク・オプション -NOMessage の［指定形式］を変更しました。
		187, 188, 190	リンク・オプション -CRc の［指定形式］，［詳細説明］，［備考］を変更しました。
		193	リンク・オプション -VECTN の［指定形式］，［詳細説明］を変更しました。
		198	リンク・オプション -CFI_ADD_Func の［指定形式］を変更しました。
		199	リンク・オプション -CFI_IGNORE_Module の［指定形式］，［詳細説明］，［使用例］を変更しました。
		211	リンク・オプション -STARt の［指定形式］を変更しました。
		228	リンク・オプション -CPu の［指定形式］を変更しました。
		243	リンク・オプション -REName の［指定形式］，［備考］を変更しました。
		244	リンク・オプション -DELete の［指定形式］を変更しました。
		245	リンク・オプション -REPlace の［指定形式］を変更しました。
		248	リンク・オプション -CHange_message の［指定形式］を変更しました。
		324	「メモリ配置領域指定（__near/__far）」の［機能］を変更しました。
		341	「コンパイラ出力セクション名の変更（#pragma section）」の［注意］を変更しました。
		351	「絶対番地配置指定（#pragma address）」の［制限］を変更しました。
		444	.DB 疑似命令の［詳細説明］を変更しました。
		445	.DB2 疑似命令の［詳細説明］を変更しました。
		494	表 6.1 を変更しました。
		499	以下のライブラリを追加しました。 rl78nm4s99.lib，rl78cm4s99.lib，rl78em4s99.lib，rl78em8s99.lib
		499, 500	「ライブラリ・ファイル命名規則」の説明を変更しました。
		502	「ライブラリ関数」の説明を変更しました。
		503	assert 関数の［詳細説明］を変更しました。
		504, 508	isblank 関数を追加しました。
		521-525	「最大幅整数型のための関数」を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	579, 586-588	snprintf 関数を追加しました。
		579, 598-600	vsnprintf 関数を追加しました。
		579	sprintf，vsprintf の「概要」を変更しました。
		580-582	printf 関数の［指定形式］,［詳細説明］を変更しました。 ［制限］を追加しました。
		584, 585	scanf 関数の［詳細説明］を変更しました。 ［制限］を追加しました。
		589-591	sprintf 関数の概要，［指定形式］,［詳細説明］を変更しました。 ［制限］を追加しました。
		593, 594	sscanf 関数の［詳細説明］を変更しました。 ［制限］を追加しました。
		595-597	vprintf 関数の［指定形式］,［詳細説明］を変更しました。 ［制限］を追加しました。
		601-603	vsprintf 関数の概要，［指定形式］,［詳細説明］を変更しました。 ［制限］を追加しました。
		609, 614	atoll 関数を追加しました。
		609, 617	strtold 関数を追加しました。
		609, 619	strtoll 関数を追加しました。
		609, 621	strtoull 関数を追加しました。
		609, 636	llabs 関数を追加しました。
		609, 637	lldiv 関数を追加しました。
		615	strtod 関数の［指定形式］を変更しました。 ［制限］を追加しました。
		616	strtof 関数の［指定形式］を変更しました。 ［制限］を追加しました。
		618	strtol 関数の［指定形式］を変更しました。
		620	strtoul 関数の［指定形式］を変更しました。
		632	abs 関数の［戻り値］を変更しました。
		634	labs 関数の［戻り値］を変更しました。
		639	memcpy 関数の［指定形式］を変更しました。
		641	strcpy 関数の［指定形式］を変更しました。
		642	strncpy 関数の［指定形式］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	643	strcat 関数の［指定形式］を変更しました。
		644	strncat 関数の［指定形式］を変更しました。
		655	strtok 関数の［指定形式］を変更しました。
		664-668	以下の関数を追加しました。 isblank，imaxabs，imaxdiv，strtoimax，_COM_strtoimax_ff，strtoumax，_COM_strtoumax_ff，snprintf，vsnprintf，atoll，_COM_atoll_f，strtold，_COM_strtold_ff，strtoll，_COM_strtoll_ff，strtoull，_COM_strtoull_ff，llabs，lldiv
		690，693-695，697，702，703，716，725-729，732	次のメッセージを追加しました。 E0511177，E0511182，E0520069，E0520117，E0520175，E0520223，E0520655，E0520744，E0520747，E0523003，E0523014，E0523038，E0523044，E0523048，E0523077，E0523078，F0520571，F0523088，W0511181，W0511183，W0520159，W0520221，W0520222，W0520240，W0520606，W0520609，W0520819，W0520966，W0520967，W0520968，W0521037，W0521039，W0521040，W0523018，W0523085，W0561143
		691，692，695-700，702，711，716，725-728，732	次のメッセージを変更しました。 E0520026，E0520039，E0520228，E0520296，E0520313，E0520393，E0520404，E0520469，E0520521，E0520643，E0520644，E0520654，E0520702，E0520749，E0520757，E0520765，E0520938，E0520965，E0520966，E0520967，E0520968，E0520969，E0520976，E0520977，E0521029，E0521030，E0521031，E0521037，E0521038，E0521039，E0521040，E0521045，E0521049，E0521051，E0521052，E0521144，E0521649，E0562135，E0562142，F0520163，F0520164，F0520182，F0520642，F0520920，W0520055，W0520083，W0520140，W0520223，W0520257，W0520513，W0520767，W0520867，W0520940，W0520951，W0521046，W0521051，W0521057，W0521072，W0521222，W0521223，W0521224，W0521297，W0561183，W0561184
		688	C0510000 を削除しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	690, 691, 693-697	次のエラーメッセージを削除しました。 E0511111, E0511112, E0511118, E0511119, E0511122, E0511125, E0511126, E0511127, E0511132, E0511136, E0511137, E0511138, E0511139, E0511140, E0511141, E0511142, E0511148, E0511155, E0511157, E0511158, E0511159, E0511160, E0511161, E0511165, E0511167, E0511173, E0511175, E0511176, E0511200, E0512001, E0520002, E0520079, E0520096, E0520103, E0520123, E0520126, E0520131, E0520133, E0520135, E0520150, E0520153, E0520157, E0520160, E0520194, E0520195, E0520196, E0520220, E0520227, E0520230, E0520239, E0520241, E0520242, E0520243, E0520244, E0520245, E0520246, E0520248, E0520249, E0520250, E0520251, E0520252, E0520255, E0520257, E0520258, E0520259, E0520262, E0520263, E0520264, E0520265, E0520266, E0520269, E0520276, E0520277, E0520278, E0520279, E0520280, E0520281, E0520282, E0520283, E0520285, E0520286, E0520287, E0520288, E0520289, E0520290, E0520291, E0520292, E0520293, E0520294, E0520297, E0520298, E0520299, E0520300, E0520302, E0520304, E0520305, E0520306, E0520307, E0520308, E0520309, E0520310, E0520311, E0520312, E0520314, E0520315, E0520316, E0520317, E0520318, E0520319, E0520320, E0520321, E0520322, E0520323, E0520326, E0520327, E0520328, E0520329, E0520330, E0520332, E0520333, E0520334, E0520335, E0520336, E0520337, E0520338, E0520339, E0520341, E0520342, E0520343, E0520344, E0520345, E0520346, E0520347, E0520348, E0520349, E0520350, E0520351, E0520352, E0520353, E0520354, E0520356, E0520357, E0520358, E0520359, E0520360, E0520363, E0520364, E0520365, E0520366, E0520367, E0520369, E0520371, E0520372, E0520373, E0520378, E0520380, E0520384, E0520386, E0520389, E0520390, E0520391, E0520392, E0520394, E0520397, E0520403, E0520405, E0520407, E0520408, E0520410, E0520412, E0520413, E0520415, E0520416, E0520417, E0520418, E0520424, E0520427, E0520429, E0520432, E0520433, E0520434, E0520435, E0520436, E0520437, E0520438, E0520439, E0520440, E0520441, E0520442, E0520443, E0520449, E0520452, E0520456, E0520457, E0520458, E0520459, E0520461, E0520463, E0520464, E0520466, E0520467, E0520468, E0520470, E0520471, E0520473, E0520475, E0520476, E0520477, E0520478, E0520481, E0520484, E0520485, E0520486, E0520487, E0520489, E0520493, E0520496, E0520498, E0520500, E0520501, E0520502, E0520503, E0520504, E0520505, E0520506, E0520507, E0520508, E0520510, E0520511, E0520515, E0520516, E0520517, E0520518, E0520519, E0520529, E0520530, E0520531, E0520532, E0520536, E0520540, E0520543, E0520544, E0520546, E0520548, E0520551, E0520555, E0520556, E0520558, E0520559, E0520598, E0520599, E0520601, E0520603, E0520604, E0520605, E0520612, E0520615, E0520616, E0520620, E0520647, E0520651, E0520656, E0520658, E0520661, E0520663, E0520664, E0520665, E0520666, E0520667, E0520668, E0520669, E0520670, E0520673, E0520674, E0520691, E0520692, E0520693, E0520694, E0520695, E0520696, E0520697, E0520698, E0520701, E0520703, E0520704, E0520706, E0520707, E0520709, E0520710, E0520711, E0520717, E0520718, E0520719, E0520724, E0520725, E0520726, E0520727, E0520728, E0520730, E0520734, E0520735, E0520742, E0520750, E0520751, E0520752, E0520753, E0520754, E0520755, E0520756, E0520758, E0520759, E0520769, E0520771, E0520772, E0520773, E0520774, E0520775, E0520776

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	697-703	次のエラーメッセージを削除しました。 E0520777, E0520779, E0520782, E0520785, E0520786, E0520787, E0520788, E0520789, E0520790, E0520791, E0520792, E0520795, E0520799, E0520800, E0520801, E0520803, E0520804, E0520805, E0520807, E0520808, E0520809, E0520810, E0520812, E0520817, E0520818, E0520819, E0520822, E0520824, E0520827, E0520828, E0520832, E0520834, E0520835, E0520840, E0520841, E0520842, E0520844, E0520847, E0520848, E0520849, E0520850, E0520851, E0520854, E0520855, E0520857, E0520864, E0520865, E0520868, E0520871, E0520872, E0520873, E0520875, E0520876, E0520877, E0520878, E0520879, E0520880, E0520881, E0520882, E0520885, E0520890, E0520891, E0520892, E0520893, E0520894, E0520896, E0520898, E0520901, E0520915, E0520916, E0520928, E0520929, E0520930, E0520934, E0520937, E0520939, E0520948, E0520952, E0520954, E0520955, E0520956, E0520960, E0520963, E0520964, E0520971, E0520972, E0520975, E0520978, E0520979, E0520980, E0520982, E0520985, E0520987, E0520988, E0520989, E0520994, E0520995, E0520996, E0520998, E0520999, E0521001, E0521006, E0521007, E0521009, E0521010, E0521011, E0521013, E0521014, E0521015, E0521017, E0521018, E0521019, E0521020, E0521021, E0521022, E0521023, E0521032, E0521034, E0521035, E0521042, E0521043, E0521044, E0521047, E0521054, E0521061, E0521065, E0521066, E0521067, E0521075, E0521076, E0521081, E0521086, E0521087, E0521088, E0521089, E0521146, E0521161, E0521163, E0521201, E0521204, E0521212, E0521227, E0521229, E0521230, E0521254, E0521255, E0521280, E0521282, E0521291, E0521292, E0521295, E0521303, E0521304, E0521311, E0521312, E0521315, E0521317, E0521318, E0521320, E0521321, E0521322, E0521323, E0521324, E0521325, E0521326, E0521327, E0521344, E0521345, E0521349, E0521350, E0521351, E0521355, E0521365, E0521372, E0521380, E0521382, E0521398, E0521403, E0521404, E0521405, E0521424, E0521425, E0521436, E0521437, E0521441, E0521442, E0521445, E0521534, E0521535, E0521536, E0521542, E0521543, E0521557, E0521558, E0521574, E0521576, E0521577, E0521583, E0521586, E0521587, E0521588, E0521589, E0521590, E0521593, E0521596, E0521597, E0521598, E0521602, E0521619, E0521620, E0521624, E0521629, E0521631, E0521632, E0521634, E0521645, E0521646, E0523042, E0523057, E0523058, E0523059, E0523066, E0523069, E0523070, E0523071, E0523072
		716	次の致命的エラーメッセージを削除しました。 F0511128, F0512003, F0520016, F0520190, F0520219, F0520542, F0520583, F0520584, F0520641, F0520869, F0520919, F0520926, F0521083, F0521151, F0521335, F0521336, F0521337, F0521338, F0523054, F0523055, F0523056, F0523073, F0523300, F0523301, F0523302
		724	次のインフォメーションメッセージを削除しました。 M0520009, M0520018, M0520111, M0520128, M0520174, M0520193, M0520237, M0520261, M0520324, M0520381, M0520399, M0520400, M0520479, M0520487, M0520534, M0520535, M0520549, M0520618, M0520652, M0520678, M0520679, M0520815, M0520831, M0520863, M0520866, M0520949, M0521348, M0521353, M0521380, M0521381, M0523009, M0523033

Rev.	発行日	改定内容	
		ページ	ポイント
1.07	2018.06.01	725-729	次のワーニングメッセージを削除しました。 W0511143，W0511144，W0511156，W0511166，W0511168，W0511169， W0511170，W0511171，W0511172，W0511179，W0519999，W0520001， W0520014，W0520019，W0520045，W0520046，W0520047，W0520054， W0520066，W0520085，W0520086，W0520101，W0520118，W0520139， W0520144，W0520147，W0520157，W0520171，W0520181，W0520185， W0520224，W0520225，W0520226，W0520232，W0520262，W0520280， W0520284，W0520296，W0520300，W0520326，W0520335，W0520368， W0520370，W0520377，W0520381，W0520382，W0520414，W0520430， W0520497，W0520512，W0520514，W0520522，W0520523，W0520533， W0520541，W0520552，W0520553，W0520554，W0520611，W0520614， W0520617，W0520650，W0520657，W0520662，W0520691，W0520692， W0520708，W0520720，W0520722，W0520723，W0520737，W0520748， W0520760，W0520780，W0520783，W0520794，W0520802，W0520806， W0520812，W0520825，W0520826，W0520829，W0520830，W0520831， W0520836，W0520837，W0520912，W0520925，W0520936，W0520941， W0520942，W0520948，W0520959，W0520961，W0520962，W0520970， W0520973，W0520984，W0520991，W0520997，W0521028，W0521030， W0521050，W0521055，W0521105，W0521145，W0521163，W0521192， W0521193，W0521194，W0521197，W0521211，W0521213，W0521218， W0531235，W0521273，W0521285，W0521290，W0521294，W0521296， W0521301，W0521302，W0521307，W0521308，W0521310，W0521316， W0521319，W0521342，W0521346，W0521361，W0521373，W0521374， W0521375，W0521386，W0521396，W0521400，W0521420，W0521427， W0521433，W0521443，W0521444，W0521546，W0521547，W0521548， W0521551，W0521553，W0521561，W0521564，W0521565，W0521566， W0521570，W0521607，W0521611，W0521632，W0521635，W0521636， W0521651，W0523042，W0523060，W0523063，W0523064，W0523120
		741	「個別オプションでの C99 規格指定時の標準ライブラリ指定」を追加しました。
		745	「コンパイラ・パッケージのバージョン」を追加しました。
1.08	2018.12.01	14	「最適化リンカ」の説明を変更しました。
		25, 90, 97, 258	コンパイル・オプション -misra_intermodule を追加しました。
		61	コンパイル・オプション -goptimize の［詳細説明］を変更しました。
		73	コンパイル・オプション -switch の［詳細説明］を変更しました。
		93	次の MISRA-C:2012 ルールを追加しました。 8.5 8.6
		136	アセンブル・オプション -goptimize の［詳細説明］を変更しました。
		162, 238, 245	リンク・オプション -LIB_RENname を追加しました。
		170	リンク・オプション -ENTry の［備考］を変更しました。
		173, 174	表 2.9 を変更しました。
		203	リンク・オプション -SHow の［詳細説明］を変更しました。
		204	リンク・オプション -SHow の［備考］を変更しました。
		243	リンク・オプション -MEMory の［備考］を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.08	2018.12.01	244	リンク・オプション -RENameの［詳細説明］,［備考］を変更しました。
		253	リンク・オプション -Total_sizeの［備考］を変更しました。
		290-305	「基本言語仕様」の構成を見直しました。
		314-320	「拡張言語仕様」の構成を見直しました。
		530, 533, 720	fpclassify 関数を追加しました。
		530, 534, 720	isfinite 関数を追加しました。
		530, 535, 720	isinf 関数を追加しました。
		530, 536, 720	isnan 関数を追加しました。
		530, 537, 720	isnormal 関数を追加しました。
		530, 538, 720	signbit 関数を追加しました。
		530, 541, 720	acosl 関数を追加しました。
		530, 544, 720	asinl 関数を追加しました。
		530, 547, 720	atanl 関数を追加しました。
		530, 550, 720	atan2l 関数を追加しました。
		530, 553, 720	cosl 関数を追加しました。
		530, 556, 720	sinl 関数を追加しました。
		530, 559, 721	tanl 関数を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.08	2018.12.01	530, 560, 721	acosh 関数を追加しました。
		530, 561, 721	acoshf 関数を追加しました。
		530, 562, 721	acoshl 関数を追加しました。
		530, 563, 721	asinh 関数を追加しました。
		530, 564, 721	asinhf 関数を追加しました。
		531, 565, 721	asinhl 関数を追加しました。
		531, 566, 721	atanh 関数を追加しました。
		531, 567, 721	atanhf 関数を追加しました。
		531, 568, 721	atanhl 関数を追加しました。
		531, 571, 721	coshl 関数を追加しました。
		531, 574, 721	sinhl 関数を追加しました。
		531, 577, 721	tanhl 関数を追加しました。
		531, 580, 721	expl 関数を追加しました。
		531, 583, 721	frexpl 関数を追加しました。
		531, 586, 721	ldexpl 関数を追加しました。
		531, 589, 721	logl 関数を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.08	2018.12.01	531	log10 関数の「概要」を変更しました。
		531	log10f 関数の「概要」を変更しました。
		531, 592, 721	log10l 関数を追加しました。
		531, 593, 722	log1p 関数を追加しました。
		531, 594, 722	log1pf 関数を追加しました。
		531, 595, 722	log1pl 関数を追加しました。
		531, 598, 722	modfl 関数を追加しました。
		532, 601, 722	fabsl 関数を追加しました。
		532, 604, 722	powl 関数を追加しました。
		532, 607, 722	sqrtl 関数を追加しました。
		532, 610, 722	ceill 関数を追加しました。
		532, 613, 722	floorl 関数を追加しました。
		532, 616, 722	fmodl 関数を追加しました。
		545	atan 関数の［詳細説明］を変更しました。
		546	atanf 関数の［詳細説明］を変更しました。
		576	tanhf 関数の［詳細説明］を変更しました。
		624, 645-648, 723	vscanf 関数を追加しました。
		624, 655-658, 723	vsscanf 関数を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.08	2018.12.01	628-631	scanf 関数の［指定形式］,［詳細説明］,［制限］を変更しました。
		638-641	sscanf 関数の［指定形式］,［詳細説明］,［制限］を変更しました。
		722	sqrt 関数の「割り込み禁止時間」を変更しました。
		722	sqrtf 関数の「割り込み禁止時間」を変更しました。
		725	strtok 関数の「.data 使用」,「.bss 使用」を変更しました。
		730	「スタック・ポインタの設定，スタック領域の初期化」の説明を変更しました。
		736, 737	「ROM 化」を削除し，「ROM イメージの作成」を追加しました。
		745, 750, 772, 783	次のメッセージを追加しました。 C0520000，C0529000，E0520079，E0562600，W0520070
		770, 788	次のメッセージを変更しました。 E0562320，W0561101
		798	「スタートアップのスタック領域初期化」を変更しました。
		804, 805	「変数や定数を各領域に配置する方法」の説明を変更しました。
		808	「プログラムを RAM 上で実行する方法」の説明を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.09	2019.11.01	11	「著作権について」の説明を変更しました。
		87	コンパイル・オプション -nest_comment の「省略時解釈」を変更しました。
		92	次の MISRA-C:2012 ルールを追加しました。 8.13 14.2 14.3
		158, 162, 170, 174, 243	リンク・オプション -ALLOW_DUPLICATE_MODULE_NAME を追加しました。
		290- 294	「C90 の処理系定義」を変更しました。
		299	(71), (72) を変更しました。
		344, 345	「コンパイラ出力セクション名の変更（#pragma section）」の説明を変更しました。
		531- 533, 600- 605, 621- 644, 648- 668, 672, 675, 775- 777	次の関数を追加しました。 scalbn, scalbnf, scalbnl, scalbln, scalblnf, scalblnl, nearbyint, nearbyintf, nearbyintl, rint, rintf, rintl, lrint, lrintf, lrintl, llrint, llrintf, llrintl, round, roundf, roundl, lround, lroundf, lroundl, llround, llroundf, llroundl, trunc, truncf, truncl, copysign, copysignf, copysignl, nan, nanf, nanl, fdim, fdimf, fdiml, fmax, fmaxf, fmaxl, fmin, fminf, fminl, isgreater, isgreaterequal, isless, islessequal, islessgreater, isunordered, va_copy

Rev.	発行日	改定内容	
		ページ	ポイント
1.10	2020.11.01	表紙	対象の CPU コアを追記しました。
		16	表 2.1 にツール使用情報ファイルを追加しました。
		22 49	コンパイル・オプション -preinclude の説明，［詳細説明］を変更しました。
		23, 69, 77, 78	コンパイル・オプション -stuff を追加しました。
		57, 58	以下の最適化項目を追加しました。 branch_chaining，align
		70	コンパイル・オプション -dbl_size の［詳細説明］を変更しました。
		164, 240, 256	リンク・オプション -VERBOSE を追加しました。
		218	リンク・オプション -USER_OPT_BYTE の［詳細説明］を変更しました。
		261	「オプションの複数回指定」に次のオプションを追加しました。 -Obranch_chaining，-Oalign
		323, 324	表 4.15 の「#pragma 指令」の書式指定を変更しました。
		346	「コンパイラ出力セクション名の変更」の［方法］を変更しました。
		368	表 4.16 の以下の組み込み関数の「機能」説明を変更しました。 __mulu，__mului，__mulsi，__mulul，__mulsl
		368, 369	表 4.16 の関数宣言を ANSI-C 形式に変更しました。
		433, 435	.SECTION 疑似命令の［指定形式］，［詳細説明］を変更しました。
		803, 819, 829- 832, 840, 843, 852	次のメッセージを追加しました。 C0580013，C0580014，C0580015，C0580016，C0580900，C0580901， C0580902，C0580903，C0580904，C0580905，C0580906，C0580907， E0550271，E0580001，E0580002，E0580003，E0580004，E0580005， E0580006，E0580007，E0580008，E0580009，E0580010，E0580011， E0580012，E0580017，E0580100，E0580104，E0580105，E0580106， E0580200，E0580201，E0580202，E0580203，E0580204，E0580300， E0580301，E0580302，E0580303，E0580305，E0580307，E0580308， E0580309，E0580310，E0580311，E0580312，E0580313，E0580314， E0580315，E0580316，E0580317，E0580318，F0580101，F0580102， F0580103，F0580399，W0520171，W0580304，W0580306

Rev.	発行日	改定内容	
		ページ	ポイント
1.11	2021.11.01	11	「概説」の説明を変更しました。
		18	「コマンド・ラインでの操作方法」の「指定形式」の説明を変更しました。
		48	コンパイル・オプション -I の［詳細説明］を変更しました。
		56	最適化項目 inline_size の説明を変更しました。
		143	アセンブル・オプション -include の［詳細説明］を変更しました。
		209	リンク・オプション -Optimize の［詳細説明］を変更しました。
		220	リンク・オプション -SECURITY_ID の［詳細説明］を変更しました。
		324	「#pragma 指令」の説明を変更しました。
		373, 376	シンボルに $ を追加、および「シンボル記述上の規則」に説明を追記しました。
		459, 463	.WEAK 疑似命令を追加しました。
		464, 470	.ALIAS 疑似命令を追加しました。
		464, 471	.TYPE 疑似命令を追加しました。
		804	「他言語で定義された変数の参照」において，参照例の誤記を訂正しました。
		808, 843, 846, 852	次のメッセージを変更しました。 E0511178，F0563430，W0511180，W0511185，W0561016，W0561017
		823	次のメッセージを追加しました。 E0550272
1.12	2022.12.01	11	「著作権について」の説明を変更しました。
		15	表 2.1 に C++ ソース・ファイルを追加しました。
		18	「コマンド・ラインでの操作方法」の説明を変更しました。
		22, 38	コンパイル・オプション -use_mda の説明，［指定形式］，［詳細説明］を変更，［注意事項］を追加しました。
		22, 40	コンパイル・オプション -lang の説明，［指定形式］，［詳細説明］，［備考］を変更しました。
		48	コンパイル・オプション -I の［詳細説明］を変更しました。
		55, 56, 58	コンパイル・オプション -O の［指定形式］，［詳細説明］，［使用例］を変更しました。
		140	アセンブル・オプション -define の［詳細説明］を変更しました。
		162, 174, 204, 205	リンク・オプション -RAM_INIT_TABLE_SECTION を追加しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.12	2022.12.01	163, 216, 222	リンク・オプション -SECURITY_OPT_BYTE を追加しました。
		163, 216, 225, 226	リンク・オプション -FLASH_SECURITY_ID を追加しました。
		163, 216, 228-230	リンク・オプション -SPLIT_SECTION を追加しました。
		163, 216, 231-233	リンク・オプション -STRIDE_DSP_MEMORY_AREA を追加しました。
		163, 216, 242	リンク・オプション -DSP_MEMORY_AREA を追加しました。
		192, 195	リンク・オプション -CRc の［詳細説明］,［使用例］を変更しました。
		211	リンク・オプション -Optimize の［詳細説明］を変更しました。
		221	リンク・オプション -OCDBG の［詳細説明］を変更しました。
		223	リンク・オプション -SECURITY_ID の［詳細説明］を変更しました。
		234	リンク・オプション -DEBUG_MONITOR の［詳細説明］を変更しました。
		274, 275	オプションの複数指定に関する内容を変更しました。
		283	再配置属性に FLASH_SECUR_ID を追加しました。
		348	「メモリ配置領域指定（__near/__far）」の「ポインタ比較」の説明を変更しました。
		446-448	.SECTION 疑似命令の［指定形式］,［詳細説明］を変更しました。
		451	.CSEG 疑似命令の［詳細説明］を変更しました。
		518	表 6.1 に以下のセクションを追加しました。 .flash_security_id
		807-809	「初期化テーブルを利用した RAM 領域セクションの初期化処理【V1.12 以降】」を追加しました。
		811-814	表 8.1 を変更しました。
		855, 861, 870	次のメッセージを変更しました。 F0520571，F0563430，W0561017
		864, 870	次のメッセージを追加しました。 W0511186，W0511187，W0561018

Rev.	発行日	改定内容	
		ページ	ポイント
1.13	2023.12.01	13-15, 他	ライブラリジェネレータに関する記述を追加しました。
		18-20	「コマンド・ラインでの操作方法」の説明を変更しました
		21, 276-286	「ライブラリジェネレータ・オプション【V1.13.00 以降】」を追加しました。
		22, 36, 39	コンパイル・オプション -use_mach を追加しました。
		22, 43	コンパイル・オプション -P の説明，［詳細説明］，［使用例］を変更しました。
		100	コンパイル・オプション -misra_intermodule に［備考］を追加しました。
		163, 211, 217	リンク・オプション -ALLOW_OPTIMIZE_ENTRY_BLOCK を追加しました。
		183	リンク・オプション -Rom の［詳細説明］を変更しました。
		191	リンク・オプション -PADDING の［詳細説明］を変更しました。
		193	リンク・オプション -CRc の［詳細説明］を変更しました。
		234	リンク・オプション -STRIDE_DSP_MEMORY_AREA の［使用例］を変更しました。
		236	リンク・オプション -DEBUG_MONITOR の［使用例］を変更しました。
		237	リンク・オプション -RRM の［詳細説明］，［備考］を変更しました。
		366, 367	「ハードウェア割り込みハンドラ（#pragma interrupt）」の［方法］を変更しました。
		370	「ソフトウェア割り込みハンドラ（#pragma interrupt_brk）」の［方法］を変更しました。
		394	表 4.16 の以下の組み込み関数の「機能」欄を変更しました。 __mulul，__mulsl
		449	「絶対式と相対式」の説明の誤記を修正しました。
		531	表 6.1 の注釈を削除しました。
		535	表 6.2 に -rrm オプションを追加しました。
		830, 831	「値が保証される汎用レジスタおよび ES/CS レジスタ」の内容を変更しました。
		831	「値が保証されるシステム・レジスタ」を削除しました。
		833	「戻り値」の説明を変更しました。
		835	「アセンブリ言語から C 言語ルーチンの呼び出し」の「レジスタ」の説明を変更しました。

Rev.	発行日	改定内容	
		ページ	ポイント
1.13	2023.12.01	837, 868	次のメッセージを変更しました。 C0519996，F0520003
		838, 860, 875, 876, 888	次のメッセージを追加しました。 C0590001，E0562114，F0593000，F0593021，F0593300，F0593302， F0593303，F0593305，F0593320，F0593321，F0593322，F0593324， F0593325，F0593326，F0593327，F0593328，F0593329，F0593330， W0591300，W0591301

CC-RL ユーザーズマニュアル

発行年月日　2014年　8月1日　Rev.1.00
　　2023年 12月1日　Rev.1.13

発行　　ルネサス エレクトロニクス株式会社
　　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

CC-RL

RENESAS

ルネサス エレクトロニクス株式会社

R20UT3123JJ0113