User's Manual

CC-RX

コンパイラ

ユーザーズマニュアル

対象リビジョン
V2.00.00 - V3.06.00

対象デバイス
RXファミリ

対象CPUコア
RXv1, RXv2, RXv3

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、RX 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-RX）について説明します。

対象者

このマニュアルは、CC-RX 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-RX 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「RX ファミリ向け C/C++ コンパイラ」CC-RX V2.00 ～ V3.06 向けのユーザーズマニュアルです。
この章では、CC-RX が行うコンパイル処理の概要と、プログラム開発の事例を説明します。

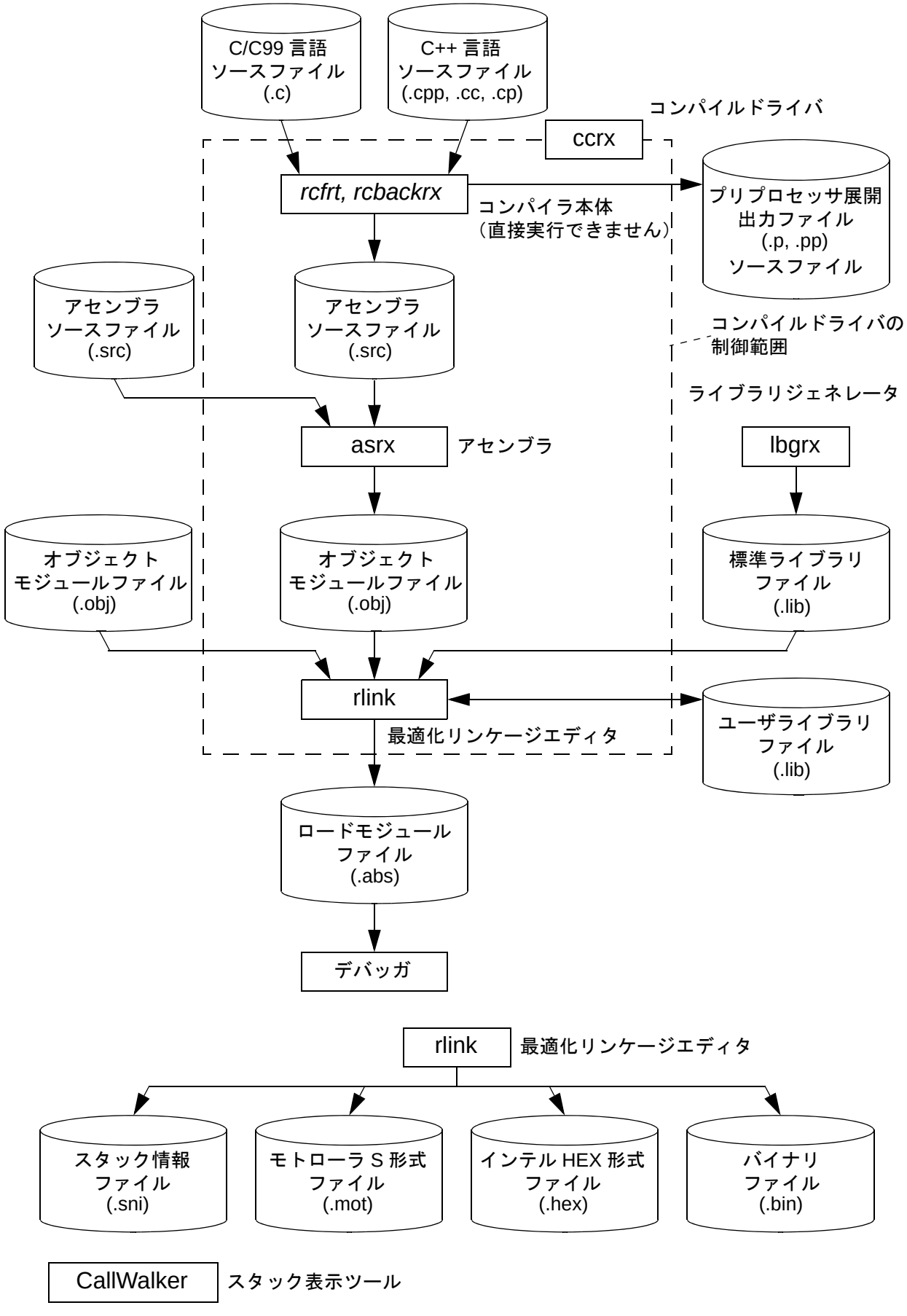
1.1 概　　要

CC-RX は、C/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。
CC-RX は、以下に示す実行ファイルで構成されています。

(1) ccrx: コンパイルドライバ
(2) asrx: アセンブラ
(3) rlink: 最適化リンケージエディタ
(4) lbgrx: ライブラリジェネレータ

以下に、ビルド・ツールの処理の流れを示します。

図 1.1　ビルド・ツールの処理の流れ



1.2 著作権について

本ソフトウェアは LLVM 及び Protocol Buffers を利用しています。
・LLVM 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。
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。

1.3 特　　長

RX ファミリ用 C/C++ コンパイラパッケージ (CC-RX) は、次の特長を備えています。

(1) **ANSI** 規格に準拠した言語仕様
C,C99,C++ 言語仕様は、ANSI 規格に準拠しています。また、従来の C 言語仕様（K&R 仕様）との両立性も備えています。

(2) 高度な最適化
コンパイラによるコード・サイズ、および速度優先の最適化を提供しています。

(3) 記述性の向上
拡張言語仕様によりプログラミングの記述性を向上させています。

(4) 高い移植性
CC-RX では単一のコンパイラですべてのマイクロコントローラをサポートしています。これにより言語仕様の統一を図り、マイクロコントローラ間の移行を容易にしています。
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。

1.4 最大値

1.4.1 コンパイラの最大値

ソースプログラムを作成する際は、この翻訳限界の範囲で作成してください。

表 1.1　　コンパイラの翻訳限界

No.	分類	項目	翻訳限界
1	起動	define オプションで指定可能なマクロ名総数	制限なし (メモリ容量に依存)
2		ファイル名の文字数	制限なし (OS に依存)
3	ソース プログラム	1 行の文字数	32768 文字
4		1 ファイルあたりのソースプログラムの行数	制限なし (メモリ容量に依存)
5		コンパイル可能なソースプログラムの総行数	制限なし (メモリ容量に依存)
6	プリプロ セッサ	#include 文のネストの深さ	制限なし (メモリ容量に依存)
7		#define 文のマクロ名総数	制限なし (メモリ容量に依存)
8		マクロ定義、マクロ呼び出しのパラメータの個数	制限なし (メモリ容量に依存)
9		マクロ名の再置き換えの数	制限なし (メモリ容量に依存)
10		条件コンパイルのネストのレベル数	制限なし (メモリ容量に依存)
11		#if, #elif 文で指定可能な演算子、被演算子の合計数	制限なし (メモリ容量に依存)

No.	分類	項目	翻訳限界
12	宣言	関数定義の個数	制限なし (メモリ容量に依存)
13		外部結合となる識別子 (外部名) の数	制限なし (メモリ容量に依存)
14		1 関数内で有効な識別子 (内部名) の数	制限なし (メモリ容量に依存)
15		基本型を修飾するポインタ、配列、および関数宣言子の数	16 個
16		配列の次元数	6 次元
17		配列・構造体のサイズ	2147483647 バイト
18	文	複文のネストの深さ	制限なし (メモリ容量に依存)
19		繰り返し文 (while 文、do 文、for 文)、選択文 (if 文、switch 文) の組み合わせによるネストの深さ	4096 レベル
20		1 関数内で記述可能な複文の数	2048 個
21		1 関数内で指定可能な goto ラベルの数	2147483646 個
22		switch 文の数	2048 個
23		switch 文のネストの深さ	2048 レベル
24		1 つの switch 文内で指定可能な case ラベルの数	2147483646 個
25		for 文のネストの深さ	2048 レベル
26	式	文字列の文字数	32766 文字
27		関数定義、関数呼び出しでパラメータの個数	2147483646 個
28		1 つの式で指定可能な演算子と被演算子の合計数	約 500 個
29	標準ライブラリ	open 関数で一度にオープンできるファイルの数	可変 *1
30	セクション	セクション名長 *2	8146 文字
31		1ファイルあたりの#pragma sectionで指定できるセクション数	2045 個
32		セクションの最大サイズ	4294967295 バイト
33	出力ファイル	アセンブリソースとして出力できる 1 行の最大文字数	8190 文字

注 1. 詳細は「8.3.2 初期設定」を参照してください。

注 2. オブジェクト生成時に用いるアセンブラの 1 行文字数の制限を受けるため、#pragma section や section オプションで指定できる長さはこれより小さくなります。

1.4.2 アセンブラの最大値

ソースプログラムを作成する際は、この翻訳限界の範囲で作成してください。

表 1.2 アセンブラの翻訳限界

	項目	翻訳限界
1	1 行文字数	32760 文字
2	シンボル長	1 行文字数 *1
3	シンボル数	制限なし (メモリ容量に依存)

	項目	翻訳限界
4	外部参照シンボル数	制限なし (メモリ容量に依存)
5	外部定義シンボル数	制限なし (メモリ容量に依存)
6	セクションの最大サイズ	0FFFFFFFFH バイト
7	セクション数	65265 個 (デバッグ情報あり)、65274 個 (デバッグ情報なし)
8	ファイルインクルード	ネストは 30 レベル
9	文字列長	1 行文字数 *1
10	ファイル名の文字数	1 行文字数 *1
11	環境変数設定文字数	2048 バイト
12	マクロ定義数	65535 個
13	サブコマンドファイルの文字数	32767 文字 *2
14	サブコマンドファイルの 1 行の文字数	2048 文字

注 1.　同じ行に指定した文字列の長さにより、これよりも小さい値となります。

注 2.　他のアセンブラオプションを含めた文字数のため、これよりも小さい値となります。

1.5 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「1.6 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「1.7 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-RX V2.06 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。

1.6 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、ANSI 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「2.5 オプション」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「4.2.3 #pragma 指令」を参照してください。

1.7 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、試用期間内では、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 版の付加機能は使用できないほか、リンクサイズに制限があります。

- リンクサイズの制限は、ROM 領域に配置されるセクション・サイズの合計が 128K バイトまでになります。
  128K バイトを超えた場合はリンカエラーになります。

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、製品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]
- 製品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、以下サービス提供の対象外となります。以下のサービスが必要な場合には、製品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、ビルド・ツールに含まれる各コマンド仕様についての詳細を説明します。

2.1 概　　要

RX ファミリ用 C/C++ コンパイラは、C 言語、C99 言語、C++ 言語やアセンブリ言語で記述したソース・プログラムから、ターゲット・システムで実行可能なファイルを生成します。
RX ファミリ用 C/C++ コンパイラでは、1 つのドライバがプリプロセッサからリンクまでのフェーズを制御します。各フェーズの処理について説明します。

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

(2) プリプロセッサ
ソース・プログラム中のプリプロセス指令の処理を行います。
-output=prep オプション指定時のみ、プリプロセス処理済みファイルを出力します。

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

(4) 最適化部
C ソース・プログラムを変換した内部データ表現に対して最適化を行います。

(5) コード生成部
内部データ表現をアセンブリ・ソース・プログラムに変換します。

(6) 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・モジュール・ファイルを生成します。

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

2.2 入出力ファイル

RX ファミリ用 C/C++ コンパイラの入出力ファイルを以下に示します。

表 2.1　　**RX** ファミリ用 **C/C++** コンパイラ用入出力ファイル

ファイル種別	拡張子	入出力	説明
C ソースプログラムファイル	.c	入力	C 言語、C99 言語で記述したソース・ファイル ユーザ作成ファイルです。
C++ ソースプログラムファイル	.cpp .cc .cp	入力	C++ 言語で記述したソース・ファイル ユーザ作成ファイルです。
インクルードファイル	任意	入力	ソース・ファイルで参照するファイル C 言語、C99 言語、C++ 言語、もしくはアセンブリ言語で記述したファイルです。 ユーザ作成ファイルです。
C プログラム用プリプロセッサ展開ファイル	.p	出力	入力 C 言語または C99 言語ソース・プログラムに対してプリプロセス処理を実行した結果を出力したファイル -output=prep オプション指定時に出力します。
C++ プログラム用プリプロセッサ展開ファイル	.pp	出力	入力 C++ 言語ソース・プログラムに対してプリプロセス処理を実行した結果を出力したファイル ASCII イメージファイル -output=prep オプション指定時に出力します。

ファイル種別	拡張子	入出力	説明
アセンブリソースプログラムファイル	.src	出力	コンパイルにより、C,C99 または C++ ソースから生成したアセンブリ言語ファイル
	.src	入力	アセンブリ言語で記述したソース・ファイル
アセンブリプログラム用リストファイル	.lst	出力	アセンブル結果の情報を持つリスト・ファイル -listfile オプション指定時に出力します。 -show オプションで出力内容を選択します。
リロケータブルオブジェクトプログラムファイル	.obj	出力	機械語情報と機械語の配置アドレスに関する再配置情報、およびシンボル情報を含んだ ELF 形式ファイル
		入力	
アブソリュートロードモジュールファイル	.abs	出力	リンク結果のオブジェクト・コードの ELF 形式ファイル ヘキサ・ファイルを出力する際の入力ファイルとなります。
リンケージリストファイル	.map	出力	リンク結果の情報を持つリスト・ファイル -list オプション指定時に出力します。 -show オプションで出力内容を選択します。
ライブラリファイル	.lib	出力	複数のオブジェクト・モジュール・ファイルが登録されたファイル
		入力	
ライブラリリストファイル	.lbp	出力	ライブラリ作成結果の情報を持つリスト・ファイル -list オプション指定時に出力します。 -show オプションで出力内容を選択します。
ライブラリバックアップファイル	.lbk	出力	ライブラリジェネレータが既に存在するライブラリファイルに上書きする場合に、上書き前の内容を保存しておくファイルです。
ヘキサ・ファイル（モトローラ S フォーマット）	.mot	出力	ロード・モジュール・ファイルをモトローラ S フォーマットに変換したファイル
ヘキサ・ファイル（インテル（拡張）HEX フォーマット）	.hex	出力	ロード・モジュール・ファイルをインテル（拡張）HEX フォーマットに変換したファイル
ヘキサ・ファイル（バイナリフォーマット）	.bin	出力	ロード・モジュール・ファイルをバイナリフォーマットに変換したファイル
スタック情報ファイル	.sni	出力	スタック情報ファイル -stack オプション指定時に出力します。
デバッグ情報ファイル	.dbg	出力	デバッグ情報ファイル -sdebug オプション指定時に出力します。
拡張子 td のファイルで指定された定義を含むオブジェクトファイル	.rti	出力	拡張子 td のファイルで指定された定義を含むオブジェクトファイル
呼び出し情報ファイル	.cal	出力	呼び出し情報ファイル CallWalker で出力します。
外部シンボル割り付け情報ファイル	.bls	出力	外部シンボル割り付け情報ファイル リンク時 -map オプション指定時に出力します。
	.bls	入力	外部シンボル割り付け情報ファイル コンパイル時 -map オプションの入力ファイルとして指定します。
ジャンプテーブルファイル（アセンブリ言語）	.jmp	出力	外部定義シンボルへ分岐するジャンプテーブルのアセンブラソースファイル -jump_entries_for_pic オプション指定時に出力します。

ファイル種別	拡張子	入出力	説明
シンボルアドレスファイル（アセンブリ言語）	.fsy	出力	外部定義シンボルをアセンブラ制御命令で記述したアセンブラソースファイル -fsymbol オプション指定時に出力します。
C++ 言語機能サポートファイル	.td,.ti,.pi,.ii	出力	C++ 言語機能をサポートするための情報ファイルです。
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。

2.3 環境変数

環境変数の一覧を以下に示します。なお、統合開発環境が環境変数を設定する場合があります。統合開発環境のユーザーズマニュアルも合わせてご覧ください。

表 2.2 環境変数

No.	環境変数	説明	設定省略時の解釈
1	path	実行ファイルの格納ディレクトリを指定します。	【V3.05 以前】省略不可
2	BIN_RX	ccrx を格納したディレクトリを指定します。	＜ccrx コマンドの格納ディレクトリ＞ 【V3.05 以前】lbgrx コマンド利用時は、省略不可
3	ISA_RX *1	命令セットアーキテクチャを選択します。 < 命令セットアーキテクチャ > RXV1 RXV2 RXV3 【V3.00.00 以降】	省略時、値は設定されません。
4	INC_RX	コンパイラのインクルードファイル格納ディレクトリを指定します。	＜ccrx コマンドの格納ディレクトリ＞ \..\include
5	INC_RXA	アセンブラのインクルードファイル格納ディレクトリを指定します。	省略時、値は設定されません。
6	TMP_RX	テンポラリファイルを作成するディレクトリを指定します。	OS の設定値
7	HLNK_LIBRARY1 HLNK_LIBRARY2 HLNK_LIBRARY3	最適化リンケージエディタが使用するデフォルトライブラリ名を指定します。library オプションで指定したライブラリを優先してリンクします。その後未解決のシンボルがある場合、1,2,3 の順にデフォルトライブラリを検索します。	省略時、値は設定されません。
8	HLNK_TMP	最適化リンケージエディタがテンポラリファイルを作成するフォルダを指定します。この環境変数の指定がない場合は、カレントフォルダにテンポラリファイルを作成します。	省略時、値は設定されません。

No.	環境変数	説明	設定省略時の解釈
9	HLNK_DIR	最適化リンケージエディタの入力ファイル格納フォルダを指定します。 input オプション、library オプションで指定したファイルの検索順序は、カレントフォルダ、HLNK_DIR 指定フォルダになります。ただし、ワイルドカードで指定したファイルは、カレントフォルダ内だけ検索します。	省略時、値は設定されません。
10	CPU_RX *1	CPU 種別を指定します。 <CPU 種別 > RX600 RX200	省略時、値は設定されません。

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

2.4 操作方法

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

なお、オプションはコマンドラインの左から右に順に取り込みます。異なる意味を持つオプションの指定を同時に指定した場合、エラーや警告がない場合は、後 (右側) に書いた方が有効です。エラーや警告が出る条件や、その結果はオプションにより異なりますので、詳しくは各オプションの説明でご確認ください。

(1) 各ツールの操作方法

(a) コンパイルドライバ（**ccrx**）

ccrx はコンパイルドライバの起動コマンドです。
本コマンド起動により、コンパイル、アセンブル、リンクを行うことができます。
- 入力ファイルの拡張子が「.s」「.src」「.S」「.SRC」のいずれかである場合、コンパイルドライバはそのファイルをアセンブリ言語ファイルと解釈して、アセンブラを起動します。
- 入力ファイルの拡張子が「.c」「.C」のいずれかである場合、コンパイルドライバはそのファイルを C 言語ソースファイルと解釈して、コンパイラを起動します。
- 入力ファイルの拡張子が「.cpp」「.CPP」「.cc」「.CC」「.cp」「.CP」のいずれかである場合、コンパイルドライバはそのファイルを C++ 言語ソースファイルと解釈して、コンパイラを起動します。
これら以外の拡張子のファイルは、C 言語ソースファイルとしてコンパイルします。

入力ファイルは複数同時に指定することができます。なお、C/C++ 言語ソースファイルを入力ファイルに複数同時に指定する場合を「一括コンパイル」と呼びます。

【コマンド記述形式】

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

[ ] : [ ] 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

(b) アセンブラ（**asrx**）

asrx は、アセンブラの起動コマンドです。

【コマンド記述形式】

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

(c) 最適化リンケージエディタ（**rlink**）
rlink は、最適化リンケージエディタの起動コマンドです。
リンク処理だけではなく、以下に挙げる機能も含んでいます。

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

【コマンド記述形式】

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

(d) ライブラリジェネレータ（**lbgrx**）
ライブラリ・ジェネレータは、標準ライブラリを生成するツールです。標準ライブラリを生成する際、任意のコンパイル・オプションを指定することができます。
lbgrx は、ライブラリジェネレータの起動コマンドです。

【コマンド記述形式】

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

(2) 操作方法例

(a) コンパイル、アセンブル、リンクを **1** コマンドで実施する場合
以下の手順すべてを 1 コマンドで実施します。

- C/C++ 言語ソースファイル（tp1.c と tp2.c）を ccrx でコンパイルする
- コンパイル後、asrx でアセンブルする
- アセンブル後、rlink でリンクして、アブソリュートファイル（tp.abs）を作成する

【コマンド記述】

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

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

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

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

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

(b) コンパイルとアセンブルを **1** コマンドで実施する場合
以下の手順を 1 コマンドで実施し、別コマンドでリンカを起動して、tp.abs を作成します。

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

【コマンド記述】

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

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

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

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

(c) コンパイル、アセンブル、リンクを各々別コマンドで実施する場合
以下の個々の手順を、それぞれ 1 コマンドで実施します。

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

【コマンド記述】

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

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

(d) アセンブルとリンクを **1** コマンドで実施する場合
以下の手順すべてを 1 コマンドで実施します。

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

【コマンド記述】

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

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

(e) アセンブルとリンクを別コマンドで実施する場合
以下の個々の手順を、それぞれ 1 コマンドで実施します。

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

【コマンド記述 1】

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

【コマンド記述 2】

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

(f) 既存ライブラリのリストファイルを作成する場合

- lib1.lib のリスト (lib1.lbp) を作成する

【コマンド記述】

```
rlink -form=library -list -library=lib1.lib
```

2.5 オプション

ここでは、RX ファミリ用 C/C++ コンパイラのオプションについて各フェーズごとに説明します。
コンパイル・フェーズ → 「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 最適化リンケージエディタ（rlink）・オプション」参照
ライブラリ生成・フェーズ→「2.5.4 ライブラリジェネレータ・オプション」参照

2.5.1 コンパイル・オプション

コンパイル・フェーズのオプションの分類と説明を以下に示します。

分類	オプション	説明
ソースオプション	-lang	ソース・ファイルをコンパイルする言語を選択します。
	-include	インクルード・ファイルの取り込み先フォルダを指定します。
	-preinclude	コンパイル単位の先頭にインクルードするファイルを指定します。
	-define	マクロ定義を指定します。
	-undefine	無効化するプリデファインド・マクロを指定します。
	-message	インフォメーションレベル・メッセージを出力します。
	-nomessage	抑止するインフォメーションレベル・メッセージ番号を指定します。
	-change_message	コンパイラ出力メッセージレベル変更します。
	-no_warning 【V2.08.00 以降】	ワーニング及びインフォメーションレベル・メッセージを抑止します。
	-file_inline_path	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】ファイル間インライン展開ファイル取り込み先フォルダの指定
	-comment	コメント（/* */）のネストを許すかどうかを選択します。
	-truncated_address_initializer 【V3.01.00 以降】	C 言語において、1, 2 バイト型の外部変数をアドレスで初期化することを許します。
	-check	M16C(R8C),H8(H8S,H8SX),SuperH ファミリ用の既存プログラムとの互換性をチェックします。
	-misra2004 【Professional 版のみ】	MISRA-C:2004 ルールによるソースチェックをします。
	-misra2012 【Professional 版のみ】【V2.04.00 以降】	MISRA-C:2012 ルールによるソースチェックをします。
	-ignore_files_misra 【Professional 版のみ】	MISRA-C:2004、MISRA-C:2012 チェック対象外のファイルを指定します。
	-check_language_extension 【Professional 版のみ】	拡張機能の使用によって部分抑止している MISRA-C:2004 ルールまたは MISRA-C:2012 ルールのチェックを有効にします。
	-misra_intermodule 【Professional 版のみ】【V3.01.00 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。

分類	オプション	説明
オブジェクトオプション	-output	出力ファイル形式を選択します。
	-noline	プリプロセッサ展開時に #line を出力しません。
	-debug	オブジェクト・ファイルにデバッグ情報を出力します。
	-nodebug	オブジェクト・ファイルにデバッグ情報を出力しません。
	-g_line 【V3.02.00 以降】	最適化時にソースデバッグ用の情報を強化します。
	-section	変更するセクション名を指定します。
	-stuff	変数のアライメントに応じたセクションに配置します。
	-nostuff	変数のアライメントに応じたセクションに配置しません。
	-instalign4	分岐先を 4 バイトで命令実行向け整合します。
	-instalign8	分岐先を 8 バイトで命令実行向け整合します。
	-noinstalign	分岐先を命令実行向け整合しません。
	-nouse_div_inst	除算、剰余算に DIV,DIVU,FDIV,DDIV 命令を使用しません。
	-create_unfilled_area 【V2.03.00 以降】	.OFFSET が作る空き領域に対してデータを出力しません。
	-stack_protector/ -stack_protector_all 【Professional 版のみ】 【V2.04.00 以降】	スタック破壊検出コードを生成します。
	-avoid_cross_boundary_prefetch 【V2.07.00 以降】	文字列操作ライブラリ関数をストリング命令に展開する際、データプリフェッチによる 4 バイト境界をまたいだ読み出しを防ぎます。
	-insert_nop_with_label 【V2.08.00 以降】	ローカルラベルおよび nop 命令を挿入します。
	-control_flow_integrity 【Professional 版のみ】 【V2.08.00 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。
リストオプション	-listfile	ソース・リスト・ファイルを出力します。
	-nolistfile	ソース・リスト・ファイルを出力しません。
	-show	ソース・リスト・ファイルの内容を指定します。

分類	オプション	説明
最適化オプション (1/2)	-optimize	最適化レベルを選択します。
	-goptimize	モジュール間最適化用付加情報を出力します。
	-speed	実行性能重視の最適化を実施します。
	-size	コード・サイズ重視の最適化を実施します。
	-loop	ループ展開の最大展開数を指定します。
	-inline	自動インライン展開行います。
	-noinline	自動インライン展開を行いません。
	-file_inline	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】ファイル間インライン展開対象ファイルの指定
	-case	switch 文のコード展開方式を選択します。
	-volatile	外部変数を volatile 化します。
	-novolatile	外部変数を volatile 化しません。
	-type_size_access_to_volatile【V3.04.00 以降】	volatile を指定した変数に、変数の型のサイズでアクセスします。
	-const_copy	const 修飾された外部変数の定数伝播を実施します。
	-noconst_copy	const 宣言された外部変数の定数伝播を実施しません。
	-const_div	整数型定数による除算および剰余算を変換します。
	-noconst_div	整数型定数による除算および剰余算を変換しません。
	-library	ライブラリ関数の実行方法を選択します。
	-scope	最適化範囲を複数に分割してコンパイルします。
	-noscope	最適化範囲を複数に分割しないでコンパイルします。
	-schedule	パイプライン処理を考慮した命令並べ替えを行います。
	-noschedule	命令並べ替えを行いません。

分類	オプション	説明
最適化オプション (2/2)	-map	外部変数アクセス最適化を行います。
	-smap	コンパイル単位内で定義された外部変数に対し、外部変数アクセス最適化を行います。
	-nomap	外部変数アクセス最適化を行いません。
	-approxdiv	浮動小数点定数除算の乗算化を行います。
	-enable_register	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】register 指定変数を優先的にレジスタ割り付け
	-simple_float_conv	浮動小数点型、整数型間の型変換を一部省略します。
	-fpu	単精度浮動小数点処理命令を使用します。
	-nofpu	単精度浮動小数点処理命令を使用しません。
	-dpfpu 【V3.01.00 以降】	倍精度浮動小数点処理命令を使用します。
	-nodpfpu 【V3.01.00 以降】	倍精度浮動小数点処理命令を使用しません。
	-tfu 【V3.01.00 以降】	三角関数演算器の利用方法を選択します。
	-tfu_version【V3.05.00 以降】	三角関数演算器のバージョンを選択します。
	-nosave_tfu【V3.05.00 以降】	割り込み関数に対して、三角関数演算器 (v2) の出力の退避・回復コードを生成しません。
	-alias	ポインタ指示先の型を考慮した最適化を実施するかどうかを選択します。
	-float_order	(無効オプション) このオプションは V2.00 以降では使用できません。指定しても意味を持ちません。 【V.1.02 以前】浮動小数点式の演算順序変更の最適化を行います。
	-branch_chaining 【V3.03.00 以降】	分岐命令のサイズを削減する最適化を実施します。
	-nobranch_chaining 【V3.03.00 以降】	分岐命令のサイズを削減する最適化を実施しません。
	-ip_optimize	大域最適化を実施します。
	-merge_files	複数ソースのコンパイル結果をひとつのオブジェクトに出力します。
	-whole_program	指定ソースファイルをプログラム全体と仮定して最適化を実施します。

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

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

ソースオプション

＜コンパイル・オプション / ソースオプション＞

ソースオプションには、次のものがあります。

- -lang
- -include
- -preinclude
- -define
- -undefine
- -message
- -nomessage
- -change_message
- -no_warning 【V2.08.00 以降】
- -file_inline_path (*) 無効オプション
- -comment
- -truncated_address_initializer 【V3.01.00 以降】
- -check
- -misra2004 【Professional 版のみ】
- -misra2012 【Professional 版のみ】【V2.04.00 以降】
- -ignore_files_misra 【Professional 版のみ】
- -check_language_extension 【Professional 版のみ】
- -misra_intermodule 【Professional 版のみ】【V3.01.00 以降】

-lang

<コンパイル・オプション / ソースオプション>

[指定形式]

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

- 省略時解釈
  拡張子が cpp、cc、cp のときには C++ 言語ソースファイルとしてコンパイルし、それ以外の場合は C（C89）言語ソースファイルとしてコンパイルします。
  ただし、拡張子が src、s の場合は本オプション指定に関わらずアセンブリ言語ファイルとして扱います。

[詳細説明]

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

[備考]

- Embedded C++ 言語仕様では、catch、const_cast、dynamic_cast、explicit、mutable、namespace、reinterpret_cast、static_cast、template、throw、try、typeid、typename、using、多重継承、仮想基底クラスをサポートしていません。これらを記述した場合、エラーメッセージを出力します。
- EC++ ライブラリを使用する場合は、必ず lang=ecpp オプションを指定してください。
- 一括コンパイル (入力ファイルに C/C++ 言語ソースファイルを複数同時に入力) では、個々の C/C++ 言語ソースファイルに異なる言語を指定することはできません。指定したい言語ごとに C/C++ 言語ソースを分け、それぞれの言語で本オプションを指定して一括コンパイルを行ってください。

-include

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-include = < パス名 >[,…]
```

［詳細説明］

- インクルードファイルの存在するパス名を指定します。
- パス名が複数ある場合にはカンマ（,）で区切って指定することができます。
- 「<」、「>」で囲まれたファイルの検索は、include オプション指定フォルダ、環境変数 INC_RX 指定フォルダの順序で行います。
- 「"」、「"」で囲まれたファイルの検索は、#include 行を記述しているファイルの格納フォルダ、include オプション指定フォルダ、環境変数 INC_RX 指定フォルダの順序で行います。
- include オプション指定フォルダが複数ある場合、パス名をコマンドラインに指定した順 (左から右の順) に検索します。

-preinclude

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-preinclude = < ファイル名 >[,…]
```

［詳細説明］

- 指定したファイルの内容をコンパイル単位の先頭に取り込みます。
- ファイル名が相対パス指定の場合は、次の順序でフォルダを検索します。
  - 【V3.01.00 以前】
    - コンパイル単位のあるフォルダ
    - include オプション指定フォルダ
    - 環境変数 INC_RX 指定フォルダ
  - 【V3.02.00 以降】
    - コンパイラを起動したフォルダ
- ファイル名が複数ある場合にはカンマ（,）で区切って指定することができます。

［備考］

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

-define

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

- ソースファイル内で記述する #define と同等の効果を得ます。
- < マクロ名 >=< 文字列 > と記述することで < 文字列 > をマクロ名として定義できます。
- サブオプションに < マクロ名 > を単独で指定した場合は、そのマクロ名が定義されたものと仮定します。(1 ではなく空の値で定義します)
- < 文字列 > には、名前または整定数を記述することができます。

［備考］

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

-undefine

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

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

［備考］

- 指定可能なプリデファインドマクロについては、「コンパイラ言語仕様」の章の「マクロ名」の項目を参照ください。
- 本オプションを複数回指定した場合、指定したすべてのマクロ名が未定義となります。

-message

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-message
```

- 省略時解釈
  インフォメーションレベル・メッセージを出力しません。

［詳細説明］

- インフォメーションレベル・メッセージを出力します。

［備考］

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

-nomessage

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

- サブオプションを省略すると、全てのインフォメーションレベル・メッセージの出力を抑止します。
- サブオプションでエラー番号を指定すると、指定したインフォメーションレベル・メッセージの出力だけを抑止し、その他のインフォメーションレベル・メッセージをすべて出力します。
- エラー番号が複数ある場合にはカンマ（,）で区切って指定することができます。
- ＜エラー番号＞-＜エラー番号＞のようにハイフン（-）で抑止するエラー番号の範囲を指定することもできます。
- エラー番号には、インフォメーションを表す「M」から始まるメッセージ番号の、末尾 (右側) の 5 桁を指定してください。
  例) インフォメーションメッセージ M0523009 の場合
  -nomessage=23009

［備考］

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

-change_message

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

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

```
change_message=warning= エラー番号
```

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

```
change_message=error= エラー番号
```

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

```
change_message=information
```

- すべてのウォーニングメッセージをインフォメーションレベルに変更します。

```
change_message=warning
```

- すべてのインフォメーションメッセージをウォーニングレベルに変更します。

```
change_message=error
```

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

［備考］

- インフォメーションレベルに変更したメッセージについては、nomessage オプション指定により出力を抑止できます。
- 本オプションを指定してアセンブラや最適化リンケージエディタのメッセージ出力を制御することはできません。

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

-no_warning 【V2.08.00 以降】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-no_warning={<num>|<num>-<num>}[,…]
```

［詳細説明］

- コンパイラが出力するワーニング及びインフォメーションレベル・メッセージの内、<num> で指定された番号のものの出力を抑止します。
  アセンブラや最適化リンケージエディタが出力するメッセージを抑止することはできません。
- 本オプションは複数回指定することができます。複数回指定した場合、全ての指定が有効になります。
- 各パラメータは、カンマ（,）で区切ります。区切り文字の前後に空白は指定できません。
- 区切り文字の前後に空白を入れた場合や、パラメータを省略した場合、数字以外を指定した場合はコンパイル・エラーとします。
- パラメータに指定できる値は次のとおりです。
  - 出力を抑止するメッセージのメッセージ番号下 5 桁を指定します。
  - 指定できる値は 0 から 99999 の範囲です。
    ただし、コンパイラが出力するワーニングまたはインフォメーションレベル・メッセージの番号の範囲（51000 ～ 54999）が有効です。
    それ以外の番号のメッセージを指定した場合や、存在しないメッセージ番号を指定した場合、その指定は無視します。
  - エラーメッセージの番号を指定した場合、その指定は無視します（抑止することはできません）。
    これは、-change_message=error で、メッセージレベルをエラーレベルに変更したメッセージについても同様です。
- -nomessage と併用することも可能です。両者で同じメッセージを重複して指定した場合も有効となります。

-file_inline_path

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-file_inline_path= < パス名 >[,…]
```

［詳細説明］

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

-comment

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

- 省略時解釈
  comment=nonest です。

［詳細説明］

- comment=nest を指定した場合、ネストしたコメントの記述を可能にします。
- comment=nonest を指定した場合、コメントのネストを認識しません。

［例］

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

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

-truncated_address_initializer 【V3.01.00 以降】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-truncated_address_initializer
```

［詳細説明］

- C 言語において、1, 2 バイト型の外部変数もしくは静的変数をアドレスで初期化する記述に対し、E0520069 エラーを出力しません。代わりに警告メッセージ W0520069 を出力します。

［備考］

- アドレスの上位バイトは切り捨てられます。このため、上位バイトが 0 ではなかった場合、元のアドレス値は保持されません。
- 1, 2 バイト型に一度キャストした後に、それより大きな型にキャストするケースはエラーになります。
- 初期化する変数がビットフィールドメンバの場合はエラーになります。
- C++ 言語、Embedded C++ 言語の場合は本オプションは無効です。

-check

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

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

［備考］

- dbl_size=4 が有効な時に、R8C, M16C ファミリ用 C コンパイラ、H8, H8S, H8SX ファミリ用 C/C++ コンパイラおよび SuperH ファミリ用 C/C++ コンパイラと浮動小数点関連の変換 / ライブラリの計算結果が異なる場合があります。
  dbl_size=4 は、本コンパイラでは、double 型および long double 型を 32 ビットにしますが、各種 R8C, M16C ファミリ用 C コンパイラ（fdouble_32）、H8, H8S, H8SX ファミリ用 C/C++ コンパイラ（double=float）および SuperH ファミリ用 C/C++ コンパイラ（double=float）では、double 型のみ 32 ビットにします。

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

-misra2004 【Professional 版のみ】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

- MISRA-C:2004 のルールチェック機能を有効にし、そのチェック対象を指定します。
- misra2004=all オプション指定時は、サポートしているすべてのルールをチェック対象とします。
- misra2004=apply=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、指定されたルール番号をチェック対象とします。
- misra2004=ignore=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、指定されたルール番号以外のルールをチェック対象とします。
- misra2004=required オプション指定時は、サポートしているルールのうち、ルールの分類が “required” になっているルールをチェック対象とします。
- misra2004=required_add=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、ルールの分類が “required” になっているルールと指定されたルール番号をチェック対象とします。
- misra2004=required_remove=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、ルールの分類が “required” になっているルールから指定されたルール番号を除いたルール番号をチェック対象とします。
- misra2004 =< ファイル名 > オプション指定時は、サポートしているルールのうち、指定されたファイル名に記載されたルール番号をチェック対象とします。ファイル名で指定されたファイル内の記述は、1 ルールを 1 行で記述します。
- ルール番号は、10 進数値およびピリオド（.）で指定してください。
- MISRA-C:2004 のチェック項目に該当した場合、次の形式でメッセージを表示します。
  ファイル名（*行番号*）: M0523028 Rule *ルール番号*: *メッセージ*
- misra2004=< ファイル名 > の指定を複数回行った場合は、最後に指定したファイルのみ有効になります。

［備考］

- misra2004 オプションは複数回指定できます。ただし、複数の種類を混在させた場合は、最後に記述した種類及びそれに連続する同じ種類の指定だけを有効とします。
  (指定例)
  ... -misra2004=ignore=2.2 -misra2004=apply=2.3
  -misra2004=required_add=4.1 -misra2004=apply=4.2
  -misra2004=apply=5.2 ...
  ignore, apply, required_add の 3 種類の misra2004 が同時に指定されていますが、最後に連続して指定した 2 つの apply の指定のみが有効になります。その結果、ルール 4.2 と 5.2 がチェック対象になります。
- サポートしていないルール番号を、各オプションの < ルール番号 > に指定した場合はエラー F0523031 となり、そこで処理を中止します。
- misra2004=< ファイル名 > で指定したルールファイルがオープンできない場合、エラー F0523029 となります。また、ルールファイルからルール番号が取り出せない場合は、エラー F0523030 となり、いずれの場合もそこで処理を中止します。

- lang オプションで cpp,c99 または ecpp が選択された場合、本オプションを無視します。
- output=prep と同時に指定した場合、本オプションを無視します。
- 本オプションでサポートしている MISRA-C:2004 のルール番号を次に示します。

[required であるルール番号]
2.2 2.3
4.1 4.2
5.2 5.3 5.4
6.1 6.2 6.4 6.5
7.1
8.1 8.2 8.3 8.5 8.6 8.7 8.11 8.12
9.1 9.2 9.3
10.1 10.2 10.3 10.4 10.5 10.6
11.1 11.2 11.5
12.3 12.4 12.5 12.7 12.8 12.9 12.10 12.12
13.1 13.3 13.4
14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9 14.10
15.1 15.2 15.3 15.4 15.5
16.1 16.3 16.5 16.6 16.9
18.1 18.4
19.3 19.6 19.8 19.11 19.14 19.15
20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12

[required ではないルール番号]
5.5 5.6
6.3
11.3 11.4
12.1 12.6 12.11 12.13
13.2
17.5
19.7 19.13

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

-misra2012 【Professional 版のみ】 【V2.04.00 以降】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-misra2012= {
              all
              | apply=< ルール番号 >[,...]
              | ignore=< ルール番号 >[,...]
              | required
              | required_add=< ルール番号 >[,...]
              | required_remove=< ルール番号 >[,...]
              | < ファイル名 >}
```

［詳細説明］

- MISRA-C:2012 のルールチェック機能を有効にし、そのチェック対象を指定します。
- misra2012=all オプション指定時は、サポートしているすべてのルールをチェック対象とします。
- misra2012=apply=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、指定されたルール番号をチェック対象とします。
- misra2012=ignore=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、指定されたルール番号以外のルールをチェック対象とします。
- misra2012=required オプション指定時は、サポートしているルールのうち、ルールの分類が “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。
- misra2012=required_add=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、ルールの分類が “mandatory” および “required” になっているルールと指定されたルール番号をチェック対象とします。
- misra2012=required_remove=< ルール番号 >[,< ルール番号 >,...] オプション指定時は、サポートしているルールのうち、ルールの分類が “required” になっているルールから指定されたルール番号を除いたルール番号をチェック対象とします。
- misra2012 =< ファイル名 > オプション指定時は、サポートしているルールのうち、指定されたファイル名に記載されたルール番号をチェック対象とします。ファイル名で指定されたファイル内の記述は、1 ルールを 1 行で記述します。
- ルール番号は、10 進数値およびピリオド（.）で指定してください。
- MISRA-C:2012 のチェック項目に該当した場合、次の形式でメッセージを表示します。
  ファイル名（*行番号*）: M0523086 Rule *ルール番号*: *メッセージ*
- misra2012=< ファイル名 > の指定を複数回行った場合は、最後に指定したファイルのみ有効になります。

［備考］

- misra2012 オプションは複数回指定できます。ただし、複数の種類を混在させた場合は、最後に記述した種類及びそれに連続する同じ種類の指定だけを有効とします。
  (指定例)
  ... -misra2012=ignore=3.1
  -misra2012=required_add=4.1 -misra2012=apply=4.2
  -misra2012=apply=5.2 ...
  ignore, apply, required_add の 3 種類の -misra2012 が同時に指定されていますが、最後に連続して指定した 2 つの apply の指定のみが有効になります。その結果、ルール 4.2 と 5.2 がチェック対象になります。
- サポートしていないルール番号を、各オプションの < ルール番号 > に指定した場合はエラー F0523031 となり、そこで処理を中止します。

- misra2012=< ファイル名 > で指定したルールファイルがオープンできない場合、エラー F0523029 となります。また、ルールファイルからルール番号が取り出せない場合は、エラー F0523030 となり、いずれの場合もそこで処理を中止します。
- lang オプションで cpp または ecpp が選択された場合、本オプションを無視します。
- output=prep と同時に指定した場合、本オプションを無視します。
- lang=c99 と同時に指定した場合、C90/C99 共通ルールは C99 の範囲でチェックします。
- #pragma などの拡張機能を用いたソースでは、ルールチェックの一部が抑止されます。
  抑止される内容は、check_language_extension オプションの項目を参照下さい。
- misra2012 オプションで表示される MISRA 診断メッセージは、change_message オプションで制御することはできません。
- 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。
- V3.02.00 以降でサポートしている MISRA-C:2012 のルール番号を次に示します。
  2.2 2.6 2.7
  3.1 3.2
  4.1 4.2
  5.1[*2] 5.2 5.3 5.4 5.5 5.6[*2] 5.7[*2] 5.8[*2] 5.9[*2]
  6.1 6.2
  7.1 7.2 7.3 7.4
  8.1 8.2 8.3[*2] 8.4 8.5[*3] 8.6[*3] 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[*1] 21.9 21.10 21.11 21.12 21.13 21.15 21.16

  (*1) MISRA C:2012 Amendment 1 に沿ってチェックします（getenv をチェックしません）。
  (*2) オプション -misra_intermodule を指定した場合、複数ファイルにまたがった解析が可能になります。
  (*3) オプション -misra_intermodule を指定した場合だけ有効になります。

［使用例］

- 次の例では、ignore, apply, required_add の 3 種類の -misra2012 が同時に指定されていますが、最後に連続して指定した 2 つの apply の指定のみが有効になります。その結果、ルール 4.2 と 5.2 がチェック対象になります。

```
-misra2012=ignore=3.1
-misra2012=required_add=4.1 -misra2012=apply=4.2
-misra2012=apply=5.2
```

-ignore_files_misra 【Professional 版のみ】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

- MISRA-C:2004 または MISRA-C:2012 のルールチェック対象外のソースファイルを指定します。

［備考］

- コマンドライン上で同一オプションを複数回指定した場合には、それぞれのオプションが有効になります。
- misra2004 または misra2012 オプションの指定がない場合は、本オプションを無視します。
- 指定されたソースファイル名が、コンパイル対象でない場合は、指定されたソースファイル名を無視します。

-check_language_extension 【Professional 版のみ】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-check_language_extension
```

［詳細説明］

- C 言語規格から独自に拡張した言語仕様が有効な場合に一部抑止される、MISRA2004 または MISRA2012 ルールチェックを有効にします。
- 本コンパイラでは、misra2004 及び misra2012 ともにデフォルトで次のチェックが抑止されます。チェックしたい場合は、misra2004 または misra2012 オプション指定時に、check_language_extension オプションを指定してください。
    - プロトタイプ宣言がない場合（MISRA-C:2004 ルール 8.1、MISRA-C:2012 ルール 8.4）で、該当関数に #pragma entry または #pragma interrupt のいずれかの指定があるとき。

［例］

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

- 関数 vfunc にはプロトタイプ宣言がありませんが、#pragma interrupt の対象なので -check_language_extension の指定がないと、ルール 8.1 のチェックメッセージが表示されません。

［備考］

- misra2004 または misra2012 オプションの指定がない場合は、本オプションを無視します。

-misra_intermodule 【Professional 版のみ】 【V3.01.00 以降】

＜コンパイル・オプション / ソースオプション＞

［指定形式］

```
-misra_intermodule=< ファイル名 >
```

- 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 < ファイル名 > 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。< ファイル名 > が存在しない場合は新規作成し、< ファイル名 > 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。
- < ファイル名 > 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。
  5.1 5.6 5.7 5.8 5.9
  8.3 8.5 8.6

［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。

```
> ccrx -isa=rxv3 -misra2012=all -misra_intermodule=test.mi a.c b.c c.c
```

［備考］

- < ファイル名 > 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、< ファイル名 > 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、< ファイル名 > 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。
- < ファイル名 > 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 < ファイル名 > 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 < ファイル名 > の情報を更新できないため、< ファイル名 > 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。
- 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。

オブジェクトオプション

＜コンパイル・オプション / オブジェクトオプション＞

オブジェクトオプションには、次のものがあります。

- -output
- -noline
- -debug
- -nodebug
- -g_line 【V3.02.00 以降】
- -section
- -stuff
- -nostuff
- -instalign4
- -instalign8
- -noinstalign
- -nouse_div_inst
- -create_unfilled_area【V2.03.00 以降】
- -stack_protector/-stack_protector_all 【Professional 版のみ】【V2.04.00 以降】
- -avoid_cross_boundary_prefetch 【V2.07.00 以降】
- -insert_nop_with_label 【V2.08.00 以降】
- -control_flow_integrity 【Professional 版のみ】【V2.08.00 以降】

-output

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

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

- 省略時解釈
  output=obj です。

［詳細説明］

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

表 2.3 サブオプション出力形式

サブオプション	出力形式	ファイル名指定省略時の拡張子
prep	プリプロセッサ展開後のソースファイル	C（C89, C99）言語ソースファイル：p C++ 言語ソースファイル：pp
src	アセンブリ・ソース・ファイル	src
obj	リロケータブル・ファイル	obj
abs	アブソリュート・ファイル	abs
hex	インテル HEX 形式ファイル	hex
sty	モトローラ S 形式ファイル	mot

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

［備考］

- 指定した形式のファイルを作成するための中間ファイルは作成されません。

-noline

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-noline
```

［詳細説明］

- プリプロセッサ展開時に #line 出力を抑止します。

［備考］

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

-debug

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-debug
```

- 省略時解釈
  -nodebug です。

［詳細説明］

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

-nodebug

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-nodebug
```

- 省略時解釈
  -nodebug です。

［詳細説明］

- nodebug オプションを指定した場合、デバッグ情報を出力しません。

-g_line 【V3.02.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-g_line
```

［詳細説明］

- -debug 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。
- 最適化を行なった場合に、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。
- デバッグ情報が増加し、ステップ実行が遅くなる場合があります。

［例］

```
ccrx a.c -isa=rxv3 -debug -g_line
```

-section

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

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

- 省略時解釈
  section=P=P,C=C,D=D,B=B,L=L,W=W です。

［詳細説明］

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

［備考］

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

-stuff

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-stuff
```

［詳細説明］

- stuff オプションを指定した場合、すべての変数をアライメント数に応じてアライメント数が 4 のセクション、2 のセクション、1 のセクションに配置します（表 2.4）。
- 【V3.01.00 以降】dpfpu オプションを指定した場合、double および long double 型の変数をアライメント数が 8 のセクションに配置します。

表 2.4 stuff オプション指定時の、各変数と出力先セクションの関係

変数の種類	変数のアライメント数	変数が所属するセクション
const 修飾変数	4	C_8[*1] C[*2]
	2	C_2
	1	C_1
初期値あり変数	4	D_8[*1] D[*2]
	2	D_2
	1	D_1
初期値なし変数	4	B_8[*1] B[*2]
	2	B_2
	1	B_1
switch 文分岐テーブル	4	W
	2	W_2
	1	W_1

注 1. 変数の型が double/long double 型で、かつ dpfpu オプションを指定した場合。

注 2. 注 1. 以外の場合。

- C、D、B は section オプションまたは #pragma section で指定したセクション名になります。
- W は section オプションで指定したセクション名になります。
- 各セクション内のデータは定義順に出力されます。

［例］

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

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

［備考］

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

-nostuff

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

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

［詳細説明］

- nostuff オプションを指定した場合、指定した < セクション種別 > に属する変数をアライメント数が 4 のセクションに配置します。
- < セクション種別 > を省略した場合は、すべてのセクション種別の変数が対象になります。
- C、D、B は section オプションまたは #pragma section で指定したセクション名になります。
- W は section オプションで指定したセクション名になります。
- 各セクション内のデータは定義順に出力されます。

［例］

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

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

［備考］

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

-instalign4

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

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

- 省略時解釈
  分岐先の命令実行向け整合を行いません。

［詳細説明］

- 分岐先の命令実行向け整合を行います。
- instalign4 を指定した場合は 4 バイト配置アドレスを命令実行向け整合します。
- 命令実行向け整合とは、-instalign4 のサブオプションで選択した種類の分岐先の命令が、アライメント数（4）の倍数であるアドレスをまたいでいる場合（*1）にだけ整合を取るものです。
- 分岐先の種類は、-instalign4 のサブオプションの指定により次の 3 種類から選択します（*2）。
- 指定なし : 関数先頭、switch 文の case および default ラベル
- inmostloop: 各最内周ループの先頭、関数先頭、switch 文の case および default ラベル
- loop: 各ループの先頭、関数先頭、switch 文の case および default ラベル
- 本オプションを選択すると、プログラムセクションのアライメント数が 1 から 4 に変わります。
- 本オプションは、分岐先命令のアドレス整合することで RX CPU の命令キューを効率よく動作させ、プログラムの実行速度向上を図るものです。
- 次の製品で速度向上を図る場合に適しています。
  RX110、RX111、RX113、RX130、RX13T、RX210、RX21A、RX220 グループ
  その他の製品については -instalign8 をご覧ください。

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

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

- 本オプションを使用しないでコンパイルして生成したオブジェクト・モジュール・ファイルや標準ライブラリとリンクした場合、リンク時に W0561322 の警告を出力しますが、動作は問題ありません。

-instalign8

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

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

- 省略時解釈
  分岐先の命令実行向け整合を行いません。

［詳細説明］

- 分岐先の命令実行向け整合を行います。
- instalign8 を指定した場合は 8 バイトでそれぞれ配置アドレスを命令実行向け整合します。
- 命令実行向け整合とは、-instalign8 のサブオプションで選択した種類の分岐先の命令が、アライメント数（8）の倍数であるアドレスをまたいでいる場合（*1）にだけ整合を取るものです。
- 分岐先の種類は、-instalign8 のサブオプションの指定により次の 3 種類から選択します（*2）。
- 指定なし : 関数先頭、switch 文の case および default ラベル
- inmostloop: 各最内周ループの先頭、関数先頭、switch 文の case および default ラベル
- loop: 各ループの先頭、関数先頭、switch 文の case および default ラベル
- 本オプションを選択すると、プログラムセクションのアライメント数が 1 から 8 に変わります。
- 本オプションは、分岐先命令のアドレス整合することで RX CPU の命令キューを効率よく動作させ、プログラムの実行速度向上を図るものです。
- CPU として RXv2 以降を搭載した製品で速度向上を図る場合に適しています。その他の製品については -instalign4 をご覧ください。

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

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

- 本オプションを使用しないでコンパイルして生成したオブジェクト・モジュール・ファイルや標準ライブラリとリンクした場合、リンク時に W0561322 の警告を出力しますが、動作は問題ありません。

［例］

- ＜C ソース＞

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

- ＜出力コード＞
  [-instalign8 を指定してコンパイルした場合]
  下記は、各関数の先頭を、8 バイトで命令実行向け整合させた場合の例です。
  8 バイトの命令実行向け整合では、対象命令が 8 バイトの境界をまたがない限り、配置アドレスを変更しないため、実際に整合がかかるのは関数 f2 のアドレスだけです。

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

-noinstalign

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-noinstalign
```

［詳細説明］

- 分岐先の命令実行向け整合を行いません。

-nouse_div_inst

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-nouse_div_inst
```

［詳細説明］

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

［備考］

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

-create_unfilled_area【V2.03.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-create_unfilled_area
```

［詳細説明］

- S レコード・ファイル (～ .mot) またはヘキサ・ファイル (～ .hex) を出力する際に、アセンブリ言語において .OFFSET 疑似命令を記述した箇所に作られる空き領域に対し、データを出力しないようにします。
- 本オプションの機能を利用するには、rlink で S レコード・ファイルおよびヘキサ・ファイルを作成するときだけではなく、ccrx コマンドまたは asrx コマンドでオブジェクト・ファイル (～ .obj) を作成するときにも指定する必要があります。

［備考］

- 本オプションを使用すると、.OFFSET 疑似命令ごとに、次のような形式のシンボル (*1) を追加します。
  ___$_<FileName>_<SectionName>_<IDNumber>s__unfilled_area
  ___$_<FileName>_<SectionName>_<IDNumber>e__unfilled_area
  ここで、<FileName> にはソースファイル名、<SectionName> にはセクション名、<IDNumber> には 1 から始まる数値が入ります。

注 1. これらの形式のシンボルは予約されており、お客様のソースコードに直接記述することはできません。

-stack_protector/-stack_protector_all 【Professional 版のみ】 【V2.04.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-stack_protector[=< 数値 >]
-stack_protector_all[=< 数値 >]
```

［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（0xFFFFFFFF 番地に向かう方向）に 4 バイトの領域を確保し、その領域に < 数値 > で指定した値を格納します。
  (2) 関数の出口で、< 数値 > を格納した 4 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。
- < 数値 > には 0 から 4294967295 までの 10 進数または 16 進数の整数値を指定します。< 数値 > 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。
  - 返却値の型は void 型のみであり、仮引数を持たない関数です。
  - 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。
  - C++ プログラム内で __stack_chk_fail 関数を定義する場合は「extern "C"」を付加してください。
  - 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 関数を定義する場合は、static を指定しないでください。
- -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。
- 以下の #pragma が指定された関数は、本オプションを指定した場合でもスタック破壊検出コードを生成しません。
  #pragma inline、#pragma inline_asm、#pragma entry、#pragma no_stack_protector

［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
    }
}

#ifdef __cplusplus
extern "C" {
#endif
void __stack_chk_fail(void)
{
    printf("stack is broken!");
    abort();
}
#ifdef __cplusplus
}
#endif
```

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

```
    .glb    _test
    .glb    ___stack_chk_fail
    .glb    _printf
    .glb    _abort
    .SECTION        P,CODE
_test:
    .STACK  _test=20
    MOV.L #00000000H, R14   ; 指定した＜数値＞ 0 をスタックの領域へ格納する
    PUSH.L R14
    SUB #0CH, R0
    MOV.L #00000000H, R14
    MOV.L #0000000BH, R15
    ADD #02H, R0, R5
L12:    ; parse_bb
    MOV.B R14, [R5+]
    ADD #01H, R14
    SUB #01H, R15
    BNE L12
L13:    ; return
    MOV.L 0CH[R0], R14      ; 関数の入口で＜数値＞を格納した位置からロードし、
    CMP #00H, R14           ; 指定した＜数値＞0 と比較する
    BNE L15                 ; 異なっている場合には、L15 へ分岐する
L14:    ; return
    RTSD #10H
L15:    ; return
    BRA ___stack_chk_fail   ; __stack_chk_fail を呼び出す

___stack_chk_fail:
    .STACK  ___stack_chk_fail=8
    SUB #04H, R0
    MOV.L #_L10, R14
    MOV.L R14, [R0]
    BSR _printf
    ADD #04H, R0
    BRA _abort

    .SECTION        L,ROMDATA,ALIGN=4
_L10:
    .byte   "stack is broken!"
    .byte   00H
    .END
```

-avoid_cross_boundary_prefetch 【V2.07.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-avoid_cross_boundary_prefetch
```

［詳細説明］

- 以下の両方の条件を満たす場合に本オプションを使用すると、コンパイラは文字列操作用ライブラリ関数を、2 つのパートからなるストリング操作命令とその他の命令のコードとして展開します。文字列の読み出し開始アドレスから 4 バイト境界のアドレスまでのデータに対して文字列操作を行うコードと、4 バイト境界のアドレスから末尾アドレスまでのデータに対して文字列操作を行うコードとが、分けて出力されます。
  - 文字列操作用ライブラリ関数 memchr()、strlen()、strcpy()、strncpy()、strcmp()、strncmp()、strcat()、または strncat() を含むコードがソースファイルに存在する
  - library=intrinsic を指定して、ライブラリ関数を展開する選択をしている

［備考］

- 本オプションは、ストリング操作命令のデータプリフェッチで 4 バイト境界をまたぐ読み出しの防止を図るものです。
- 本オプションを選択すると、文字列操作用ライブラリ関数 memchr()、strlen()、strcpy()、strncpy()、strcmp()、strncmp()、strcat()、strncat() を library=intrinsic を指定してコンパイルした際のコードサイズが増加します。
- 本オプションを使用すると、ライブラリジェネレータは文字列操作用ライブラリ関数（memchr()、strlen()、strcpy()、strncpy()、strcmp()、strncmp()、strcat()、または strncat()）を、2 つのパートからなるストリング操作命令とその他の命令のコードとして生成します。文字列の読み出し開始アドレスから 4 バイト境界のアドレスまでのデータに対して文字列操作を行うコードと、4 バイト境界のアドレスから末尾アドレスまでのデータに対して文字列操作を行うコードとなります。

-insert_nop_with_label 【V2.08.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-insert_nop_with_label=< ファイル >,< 行番号 >,< ラベル >
```

- 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、-debug が有効になります。
- 本機能は CS+ または e^{2}studio 経由での使用が前提であり、ユーザは直接使用しません。

-control_flow_integrity 【Professional 版のみ】 【V2.08.00 以降】

＜コンパイル・オプション / オブジェクトオプション＞

［指定形式］

```
-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/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、不正な間接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 __control_flow_integrity(void *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/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 最適化リンケージエディタ（rlink）・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。
  - C++ プログラム内で __control_flow_chk_fail 関数を定義する場合は「extern "C"」を付加してください。
  - pic オプションを同時に指定した場合、エラーとなります。

［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＞
  -isa=rxv2 -output=src -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
  BRA _abort
_func1:
  .STACK _func1=4
  MOV.L #_glb, R14
  MOV.L [R14], R15
  ADD #01H, R15
  MOV.L R15, [R14]
  RTS
_func2:
  .STACK _func2=4
  MOV.L #_glb, R14
  MOV.L [R14], R15
  SUB #01H, R15
  MOV.L R15, [R14]
  RTS
_main:
  .STACK _main=8
  PUSH.L R6
  MOV.L #_pf, R6
  MOV.L [R6], R1
  BSR ___control_flow_integrity ; チェック関数の呼び出し
  MOV.L [R6], R14
  JSR R14 ; func1 関数の間接呼び出し
  BSR _func2 ; func2 関数の直接呼び出し
  RTSD #04H, R6-R6
  .SECTION D,ROMDATA,ALIGN=4
_pf:
  .lword _func1
  .SECTION B,DATA,ALIGN=4
_glb:
  .blkl 1
  .END
```

リストオプション

＜コンパイル・オプション / リストオプション＞

リストオプションには、次のものがあります。

- -listfile
- -nolistfile
- -show

-listfile

＜コンパイル・オプション / リストオプション＞

［指定形式］

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

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

［詳細説明］

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

［備考］

- 本オプションを指定してリンケージリストを出力することはできません。
- リンケージリストを出力するには、lnkcmd オプションにより最適化リンケージエディタの list オプションを指定してください。
- コンパイラが出力する情報は、ソースリストに書き込まれます。
- ソース・リスト・ファイルのフォーマットについては、「3.1 アセンブル・リスト・ファイル」を参照ください。
- < パス名 > を使用する場合は、パス名に該当するフォルダはあらかじめ作成しておいてください。存在しない場合は、listfile には < パス名 > の代わりに < ファイル名 > が選択されたものとみなします。

-nolistfile

＜コンパイル・オプション / リストオプション＞

［指定形式］

```
-nolistfile
```

［詳細説明］

- ソースリストファイルを出力しません。

-show

＜コンパイル・オプション / リストオプション＞

［指定形式］

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

［詳細説明］

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

表 2.5 サブオプション指定一覧

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

［備考］

- 本オプションは listfile オプション指定時のみ有効です。
- コンパイラが出力する情報は、ソースリストに書き込まれます。ソースリストファイルのフォーマットについては、「3.1 アセンブル・リスト・ファイル」を参照ください。

最適化オプション

＜コンパイル・オプション / 最適化オプション＞
最適化オプションには、次のものがあります。

- -optimize
- -goptimize
- -speed
- -size
- -loop
- -inline
- -noinline
- -file_inline (*) 無効オプション
- -case
- -volatile
- -novolatile
- -type_size_access_to_volatile【V3.04.00 以降】
- -const_copy
- -noconst_copy
- -const_div
- -noconst_div
- -library
- -scope
- -noscope
- -schedule
- -noschedule
- -map
- -smap
- -nomap
- -approxdiv
- -enable_register (*) 無効オプション
- -simple_float_conv
- -fpu
- -nofpu
- -dpfpu 【V3.01.00 以降】
- -nodpfpu 【V3.01.00 以降】
- -tfu 【V3.01.00 以降】
- -tfu_version【V3.05.00 以降】
- -nosave_tfu【V3.05.00 以降】
- -alias
- -float_order (*) 無効オプション
- -branch_chaining 【V3.03.00 以降】
- -nobranch_chaining 【V3.03.00 以降】
- -ip_optimize
- -merge_files
- -whole_program

-optimize

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

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

- 省略時解釈
  -optimize=2 です。

［詳細説明］

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

［備考］

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

-goptimize

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-goptimize
```

［詳細説明］

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

-speed

<コンパイル・オプション / 最適化オプション>

[指定形式]

```
-speed
```

- 省略時解釈
  サイズ重視の最適化を実施します。

[詳細説明]

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

[備考]

- speed オプションを指定した場合、optimize オプションの指定により、以下オプションが指定されているとみなします。
- optimize オプションの最適化レベルは、コンパイルオプション以外にも細かい最適化の調整を含みます。異なる最適化レベル同士で、下記の表のコンパイルオプションを合わせても出力コードは一致しません。

最適化項目	optimize=0 optimize=1	optimize=2	optimize=max
ループ展開	loop=1	loop=2	loop=8
インライン展開	noinline	inline=100	inline=250
定数除算の乗算化	const_div	const_div	const_div
命令並び替え	noschedule	schedule	schedule
const 修飾変数の定数伝播	noconst_copy	const_copy	const_copy
最適化範囲分割	scope	scope	noscope
外部変数アクセス最適化	nomap	nomap	map 注 nomap 注
ポインタ指示先の型を考慮した最適化	alias=noansi	alias=noansi	alias=ansi
分岐命令のサイズを削減する最適化	nobranch_chaining	nobranch_chaining	nobranch_chaining

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

-size

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-size
```

- 省略時解釈
  サイズ重視の最適化を実施します。

［詳細説明］

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

［備考］

- size オプションを指定した場合、optimize オプションの指定により、以下オプションが指定されているとみなします。ただし、以下オプションを明示的に指定した場合は指定したオプションが有効になります。
- optimize オプションの最適化レベルは、コンパイルオプション以外にも細かい最適化の調整を含みます。異なる最適化レベル同士で、下記の表のコンパイルオプションを合わせても出力コードは一致しません。

最適化項目	optimize=0 optimize=1	optimize=2	optimize=max
ループ展開	loop=1	loop=1	loop=1
インライン展開	noinline	noinline	inline=0
定数除算の乗算化	noconst_div	noconst_div	noconst_div
命令並び替え	noschedule	schedule	schedule
const 修飾変数の定数伝播	noconst_copy	const_copy	const_copy
最適化範囲分割	scope	scope	noscope
外部変数アクセス最適化	nomap	nomap	map 注 nomap 注
ポインタ指示先の型を考慮した最適化	alias=noansi	alias=noansi	alias=ansi
分岐命令のサイズを削減する最適化	nobranch_chaining	branch_chaining	branch_chaining

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

-loop

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-loop[=< 数値 >]
```

- 省略時解釈
  loop=2 です。

［詳細説明］

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

［備考］

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

-inline

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-inline[=< 数値 >]
```

- 省略時解釈
  inline=100 です。

［詳細説明］

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

［備考］

- #pragma inline を指定した関数は、本オプションの指定に関わらず、展開を試みます。inline 関数指定子 (C99) は、本オプションが有効な場合にだけ効果を持ちます。
- 確実に関数をインライン展開したい場合は、#pragma inline を関数に指定してください。
- 本オプションの選択もしくは inline 指定子を関数に指定しても、コンパイラで効率が悪くなると判断したときは、インライン展開を行わないことがあります。

-noinline

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noinline
```

［詳細説明］

- 関数の自動インライン展開を行いません。

［備考］

- #pragma inline を指定した関数は、本オプションの指定に関わらず、展開を試みます。inline 関数指定子 (C99) は、本オプションが無効な場合にだけ効果を持ちます。

-file_inline

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-file_inline=< ファイル名 >[,…]
```

［詳細説明］

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

［備考］

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

-case

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

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

- 省略時解釈
  case=auto です。

［詳細説明］

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

［備考］

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

-volatile

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-volatile
```

［詳細説明］

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

［備考］

- 本オプションで各変数に付加された volatile 宣言は、RX 用のデバッグツールでは表示されません。

-novolatile

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-novolatile
```

［詳細説明］

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

-type_size_access_to_volatile【V3.04.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-type_size_access_to_volatile
```

- 省略時解釈
  volatile を指定した変数に、変数の型のサイズとは異なるサイズでアクセスすることがあります。

［詳細説明］

- volatile を指定した変数に、変数の型のサイズでアクセスします。
- -volatile オプションによる volatile 指定にも作用します。
- 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、警告を出力しません。
- 本オプションによって変更できるのは、volatile を指定した変数に関する意味規則の内、アクセス幅だけです。その他の意味規則（アクセス回数など）は変わりません。
- 本オプションを指定していない場合に型のサイズでアクセスしたい場合は、変数の宣言時に __evenaccess キーワードを指定してください。

［備考］

- 本オプション指定におけるアクセス保証対象は４バイト以下のスカラ型です。

［使用例］

```
> ccrx a.c -isa=rxv3 -type_size_access_to_volatile
```

-const_copy

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-const_copy
```

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

［詳細説明］

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

［備考］

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

-noconst_copy

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noconst_copy
```

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

［詳細説明］

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

［備考］

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

-const_div

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-const_div
```

- 省略時解釈
  speed オプションを指定した場合は const_div です。

［詳細説明］

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

［備考］

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

-noconst_div

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noconst_div
```

- 省略時解釈
  size オプションを指定した場合は noconst_div です。

［詳細説明］

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

［備考］

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

-library

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

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

- 省略時解釈
  library=intrinsic です。

［詳細説明］

- -library=function を指定した場合、全てのライブラリ関数の呼び出しは、標準ライブラリが提供する各種サブルーチンの呼び出しになります。
- -library=intrinsic を指定した場合、次のライブラリ関数の呼び出しが、対応する機能を持つ RX 命令に置き換わります。
  - abs
  - fabsf/fabs[*1]/fabsl[*1]
  - sqrtf[*2]/sqrt[*3]/sqrtl[*3]
  - memchr/strlen/strcpy/strncpy/strcmp/strncmp/strcat/strncat

注 1. -dbl_size=4 を指定した場合、または -dpfpu を指定した場合。

注 2. -isa オプションに rxv1 以外を指定し、かつ -fpu を指定した場合。

注 3. -isa オプションに rxv1 以外を指定し、かつ -fpu -dbl_size=4 を指定した場合。または -dpfpu を指定した場合。

［備考］

- 命令に置き換わったライブラリ関数の呼び出しは、変数 errno の内容を変更しません。

-scope

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-scope
```

- 省略時解釈
  optimize オプションによって省略時解釈が変わります。
  optimize=max オプションの場合は noscope と解釈します。
  optimize=max 以外の場合は scope と解釈します。

［詳細説明］

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

-noscope

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noscope
```

- 省略時解釈
  optimize オプションによって省略時解釈が変わります。
  optimize=max オプションの場合は noscope と解釈します。
  optimize=max 以外の場合は scope と解釈します。

［詳細説明］

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

-schedule

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-schedule
```

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

［詳細説明］

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

-noschedule

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-noschedule
```

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

［詳細説明］

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

-map

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-map[=< ファイル名 >]
```

- 省略時解釈
  nomap です。ただし、output オプションに abs、hex、または sty のいずれかを指定し、かつ optimize=max オプションを指定した場合は map です。

［詳細説明］

- 外部変数アクセス最適化を行います。
- 最適化リンケージエディタが生成する外部シンボル割り付け情報を元にベースアドレスを設定し、外部変数もしくは静的変数のアクセスをベースアドレス相対で行うコードを生成します。
- map オプションによる外部変数アクセス最適化を使用する場合は、output オプションの指定により使い方が異なります。
- ［output=abs、output=sty または output=hex の場合］
  map のみ指定してください。コンパイラが自動的にコンパイル・リンクを 2 回行い、外部シンボル割り付け情報を元にベースアドレスを設定したコード生成を行います。なお、output=abs、output=sty または output=hex と、optimize=max を同時に指定した場合、map が暗黙に指定されます。
- ［output=obj の場合］
  ソースファイルを本オプションを指定しないで一度コンパイルし、最適化リンケージエディタでのリンク時に map=< ファイル名 > を指定して外部シンボル割り付け情報ファイルを作成し、再度 ccrx に map=< ファイル名 > を指定してコンパイルしてください。

［例］

- ＜C ソース＞

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

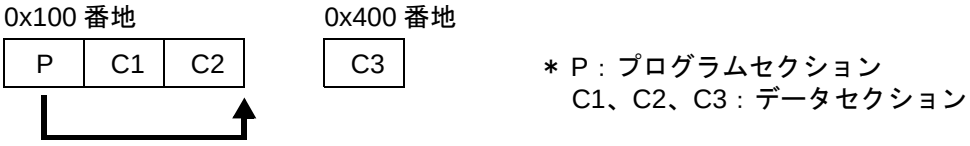
- ＜出力コード＞

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

［備考］

- 外部変数もしくは静的変数の定義順を変更した場合は、外部シンボルアドレス情報ファイルを生成し直す必要があります。map オプション以外で 1 回目のコンパイル時に指定したオプションと異なるオプションを指定した場合と、関数内の処理を追加した場合は、動作は保証しません。これらの場合は必ず外部シンボルアドレス情報ファイルを生成し直してください。

- C/C++ ソースをコンパイルする場合のみ適用されます。コンパイル時に output=src を用いて作成、またはアセンブリ言語で記述されたプログラムには適用されません。
- map オプションと smap オプションを同時に指定した場合は、map オプションが有効となります。
- プログラムセクションの次に連続してデータセクションを配置すると、外部変数アクセス最適化が無効になる、あるいは、最適化が十分に機能しない場合があります。
- 最適化を最大限に機能させるためには、連続して複数のセクションを配置させる場合、プログラムセクションを末尾に配置してください。
- 以下に具体例を示します。



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

-smap

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-smap
```

［詳細説明］

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

［例］

- ＜C ソース＞

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

- ＜出力コード＞

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

［備考］

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

-nomap

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nomap
```

- 省略時解釈
  nomap です。ただし、output オプションに abs、hex、または sty のいずれかを指定し、かつ optimize=max オプションを指定した場合は map です。

［詳細説明］

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

［例］

- ＜C ソース＞

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

- ＜出力コード＞

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

-approxdiv

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-approxdiv
```

- 省略時解釈
  浮動小数点定数除算を、定数の逆数の乗算に変換しません。

［詳細説明］

- 変数 ÷ 定数 という式があるとき、変数 ×（定数の逆数）に変換したコードを生成します。

［備考］

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

-enable_register

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-enable_register
```

［詳細説明］

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

-simple_float_conv

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-simple_float_conv
```

［詳細説明］

- 浮動小数点型の型変換処理の一部を省略します。
- 本オプション選択時は、次の浮動小数点の型変換を行う生成コードが変化します。
  a) 32bit 浮動小数点型から符号無し整数型への変換
  b) 符号無し整数型から 32bit 浮動小数点型への変換
  c) 32bit 浮動小数点型を経由した、整数型から 64bit 浮動小数点型への変換

［例］

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

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

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

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

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

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

オプション非指定時：
 _func2:
         BTST    #31,R1
         BEQ     L15
         SHLR    #1,R1,R14
         AND     #1,R1
         OR      R14,R1
         ITOF    R1,R1
         FADD    R1,R1
         BRA     L16
 L15:
         ITOF    R1,R1
 L16:
         RTS

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

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

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

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

［備考］

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

-fpu

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-fpu
```

- 省略時解釈
  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。
  CPU に RX200 を選択（*2）した場合は nofpu、それ以外は fpu です。

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

［詳細説明］

- fpu を指定した場合、単精度浮動小数点処理命令を使用したコード生成を行います。

［備考］

- 単精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。
- CPU として RX200 が選択されている場合、fpu を指定するとエラーになります。

-nofpu

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nofpu
```

- 省略時解釈
  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。
  CPU に RX200 を選択（*1）した場合は nofpu、それ以外は fpu です。

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

［詳細説明］

- nofpu を指定した場合、単精度浮動小数点処理命令を使用しないコード生成を行います。

-dpfpu 【V3.01.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-dpfpu
```

- 省略時解釈
  nodpfpu です。

［詳細説明］

- dpfpu を指定した場合、倍精度浮動小数点処理命令を使用したコード生成を行います。

［備考］

- 倍精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。
- CPU（*1）が選択されている場合及び ISA（*2）として RXv1 または RXv2 が選択されている場合、dpfpu を指定するとエラーになります。
- nofpu が指定されている場合、dpfpu を指定するとエラーになります。

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

-nodpfpu 【V3.01.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nodpfpu
```

- 省略時解釈
  nodpfpu です。

［詳細説明］

- nodpfpu を指定した場合、倍精度浮動小数点処理命令を使用しないコード生成を行います。

-tfu 【V3.01.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-tfu=intrinsic[,mathlib]
```

［詳細説明］

- -tfu=intrinsic を指定した場合は、次の三角関数演算器を利用する組み込み関数を使用できるようになります。

  - __sincosf
  - __atan2hypotf
  - __init_tfu（初期化用）
  - __sincosfx【V3.05.00 以降】
  - __sinfx【V3.05.00 以降】
  - __cosfx【V3.05.00 以降】
  - __atan2fx【V3.05.00 以降】
  - __hypotfx【V3.05.00 以降】
  - __atan2hypotfx【V3.05.00 以降】

  これらの組み込み関数の詳細は 4.2.6 組み込み関数を参照してください。

- -tfu=intrinsic,mathlib を指定した場合は組み込み関数に加えてさらに、数学ライブラリ関数の呼び出しも、三角関数演算器を利用するコードに置き換えます。置き換えの対象となる数学ライブラリ関数は次のとおりです。

  - sinf / sin[*1] / sinl[*1]
  - cosf / cos[*1] / cosl[*1]
  - atan2f / atan2[*1] / atan2l[*1]
  - hypotf / hypot[*1] / hypotl[*1]
  - asinf [*2] / asin [*1*2] / asinl [*1*2] 【V3.02.00 以降】
  - acosf [*2] / acos [*1*2] / acosl [*1*2] 【V3.02.00 以降】
  - atanf / atan [*1] / atanl [*1] 【V3.02.00 以降】
  - tanf / tan [*1] / tanl [*1] 【V3.02.00 以降】

  *1) -dbl_size=4 を指定した場合。
  *2) -isa=rxv2|rxv3 かつ -fpu の場合。

- 【V3.05.00 以降】同時に -tfu_version=v2 を指定した場合、#pragma interrupt に指定する割り込み仕様 tfu および no_tfu を使用できるようになります。

［備考］

- 【V3.05.00 未満】三角関数演算器を利用した演算処理は、リエントラントではありません。
- 【V3.05.00 以降】三角関数演算器を利用した演算処理は、-tfu_version=v1 を指定した場合はリエントラントではありません。-tfu_version=v2 を指定した場合、デフォルトはリエントラントです。ただし、次のいずれかに該当する場合はリエントラントではありません。

  - -nosave_tfu を指定した場合。詳細は「2.5.1 コンパイル・オプション」の -nosave_tfu の説明を参照してください。
  - 三角関数演算器を使用するいずれかの割り込み関数で、#pragma interrupt に割り込み仕様 no_tfu を指定した場合。詳細は「4.2.3 #pragma 指令」の #pragma interrupt の項目を参照してください。
- -tfu=intrinsic,mathlib 指定による数学ライブラリ関数の置き換えは関数呼び出しコード自体を置き換えるものであり、ライブラリ内のコードは変わりません。ポインタを使った間接呼び出しがされた場合はライブラリ関数が呼び出されるため、三角関数演算器を利用することができません。
- 数学ライブラリ関数の呼び出しが三角関数演算器を利用したコードに置き換わった場合、変数 errno の内容は変更されません。
- 三角関数演算器を利用した場合と利用しない場合では、演算精度が異なります。
- TFUv1 を使用している場合、三角関数演算器を利用する前にスタートアッププログラム等で組み込み関数 __init_tfu() を用いて演算器を初期化してください。初期化を行わなかった場合の動作は保証しません。
- TFUv2 を使用している場合、演算器の初期化は不要です。組み込み関数 __init_tfu を呼び出すとエラーになります。
- 三角関数演算器を搭載していないデバイスでは本オプションは指定しないでください。

-tfu_version【V3.05.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-tfu_version={v1|v2}
```

- 省略時解釈
  -tfu オプションを指定していない場合、未指定。-tfu オプションを指定している場合、v1。

［詳細説明］

- 三角関数演算器のバージョンを選択します。v1 を指定した場合 TFU v1 が、v2 を指定した場合 TFU v2 が選択されます。
- -tfu オプションが未指定の場合に本オプションを指定するとエラー (E0511152) になります。

-nosave_tfu【V3.05.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nosave_tfu
```

- 省略時解釈
  すべての割り込み関数に対して、三角関数演算器 (v2) の出力の退避・回復コードを生成します。

［詳細説明］

- すべての割り込み関数に対して、三角関数演算器 (v2) の出力の退避・回復コードを生成しなくなります。その結果、三角関数演算器 (v2) を利用した演算処理はリエントラントではなくなります。
- -tfu_version=v2 を指定していない場合に本オプションを指定するとエラーになります。

［備考］

- 本オプションの省略時に生成される退避・回復コードは、#pragma interrupt に割り込み仕様 tfu を選択したときに生成されるコードと同じものです。実際の退避・回復コードは、「4.2.3 #pragma 指令」の #pragma interrupt の項目を参照してください。

-alias

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

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

- 省略時解釈
  alias=noansi です。optimize=max の場合は alias=ansi となります。

［詳細説明］

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

［例］

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

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

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

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

```
_func:
    MOV.L       #_n,R4
    MOV.L       #1,[R4]     ;  n = 1;
    MOV.W       #2,[R1]     ;  *ps = 2;
    MOV.L       #_x,R4
    MOV.L       #1,[R4]     ;  (B) n = 1 で代入に使用した値を再利用する
    RTS
```

［備考］

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

-float_order

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-float_order
```

- 省略時解釈
  浮動小数点演算式の演算順序変更の最適化を行いません。

［詳細説明］

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

-branch_chaining 【V3.03.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-branch_chaining
```

［詳細説明］

- 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。
- 本オプションの省略時解釈は、optimize オプションと speed, size オプションの指定に従います。
- 詳細は、speed, size オプションを参照してください。

［備考］

- コードサイズは小さくなりますが、実行速度が低下します。
- 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。
- -speed オプションを指定した場合は本オプションを無視します。
- -optimize=0 または 1 を指定した場合は本オプションを無視します。

-nobranch_chaining 【V3.03.00 以降】

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-nobranch_chaining
```

［詳細説明］

- 分岐命令のサイズを削減する最適化を抑止します。
- 本オプションの省略時解釈は、optimize オプションと speed, size オプションの指定に従います。
- 詳細は、speed, size オプションを参照してください。

-ip_optimize

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-ip_optimize
```

［詳細説明］

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

［例］

例 1.

- ＜C ソース＞

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

- <ip_optimize の指定がない場合のアセンブリ出力 >

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

_func2:
  MOV.L #_x, R1
  ADD #04H, R1, R2
  BRA __$func1
```

- <ip_optimize 指定時のアセンブリ出力 >

```
; -optimize=2 -size

__$func1:
  MOV.L #00000000H, [R1]
  MOV.L #00000001H, [R2]
  MOV.L #00000000H, R1
  RTS

_func2:
  MOV.L #_x, R1
  ADD #04H, R1, R2
  BRA __$func1
```

例 2.

- ＜C ソース＞

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

- <ip_optimize の指定がない場合のアセンブリ出力 >

```
; -optimize=2 -size

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

- <ip_optimize 指定時のアセンブリ出力 >

```
; -optimize=2 -size

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

［備考］

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

-merge_files

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-merge_files
```

［詳細説明］

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

［例］

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

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

［備考］

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

-whole_program

＜コンパイル・オプション / 最適化オプション＞

［指定形式］

```
-whole_program
```

［詳細説明］

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

［備考］

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

［例］

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

[whole_program 指定なし ]
関数 g() は関数 func() を呼び出す可能性があるので、(2) の実行で変数 a は書き換わるとみなし、(3) では a
の値を読み出すコードを生成します。
_func:
        PUSH.L R6
        MOV.L  #__$a$1,R6
        MOV.L  [R6],R14
        ADD    #1,R14
        MOV.L  R14,[R6]     ; (1)
        BSR    _g           ; (2)
        MOV.L  [R6],R1      ; (3)
        RTSD   #4,R6-R6

[whole_program 指定あり ]
関数 g() は関数 func() 呼び出さないとみなすため、(2) の実行で変数 a は不変と判断します。これを利用し、
(3) では a の値を読み出さず、代わりに (1) で a に書き込んだ値を使用するコードを生成します。
_func:
        PUSH.L R6
        MOV.L  #__$a$1,R14
        MOV.L  [R14],R6
        ADD    #1,R6
        MOV.L  R6,[R14]     ; (1)
        BSR    _g           ; (2)
        MOV.L  R6,R1        ; (3)
        RTSD   #4,R6-R6
```

マイコンオプション

＜コンパイル・オプション / マイコンオプション＞

マイコンオプションには、次のものがあります。

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

-isa

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-isa={ rxv1 | rxv2 | rxv3 }
```

- 省略時解釈
  環境変数 ISA_RX の内容に従います。

［詳細説明］

- オブジェクトファイルを生成する際に用いる命令セットアーキテクチャを指定します。

［備考］

- -nofpu と -fpu のいずれも選択されていない場合に本オプションを指定すると、-fpu が自動的に選択されます。
- -cpu、環境変数 CPU_RX 及び環境変数 ISA_RX のいずれも設定されていない場合に本オプションを省略すると、エラーとなります。
- -cpu と本オプションを同時に指定するとエラーになります。

-cpu

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  環境変数 CPU_RX に従います。

［詳細説明］

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

［備考］

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

【注意】

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

-endian

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  endian=little です。

［詳細説明］

- endian=big を指定した場合、データのバイト並びが big endian になります。
- endian=little を指定した場合、データのバイト並びが little endian になります。
- データのバイト並びは #pragma endian 指令でも指定できます。本オプションと #pragma の両方が指定された場合には、#pragma による指定を優先します。

-round

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  round=nearest です。

［詳細説明］

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

［備考］

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

-denormalize

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  denormalize=off です。

［詳細説明］

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

［備考］

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

-dbl_size

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  dpfpu を指定した場合、dbl_size=8 です。それ以外の場合、dbl_size=4 です。

［詳細説明］

- double 型および long double 型をどのように扱うかを制御します。
  dbl_size=4 を指定した場合、単精度浮動小数点型として扱います。
  dbl_size=8 を指定した場合、倍精度浮動小数点型として扱います。

［備考］

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

-int_to_short

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-int_to_short
```

- 省略時解釈
  ソースファイル内の int を short に、unsigned int を unsigned short に置換せずコンパイルを行います。

［詳細説明］

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

［備考］

- limits.h の INT_MAX、INT_MIN、および UINT_MAX は本オプションの変換対象外となります。
- C++ および EC++ コンパイル時は、本オプションは無効になります。C++、EC++ プログラム内から C プログラムを参照する可能性がある外部名に対して W0523041 を出力します。
- C 標準ヘッダをインクルードしたファイルを int_to_short オプションを指定して C++ または EC++ コンパイルした場合も、W0523041 が出力されることがあります。この場合は動作には問題ありませんので無視してください。
- C と C++（EC++）との間で共通にアクセスするデータは、int 型ではなく long 型または short 型で宣言してください。
- 本オプション有効時は、標準ライブラリの scanf 等の書式付き入力関数の呼び出しでは、%d および %u 変換で用いる引数には、必ず long および unsigned long 型変数のアドレスをそれぞれ渡してください。long を含まない int 型や unsigned 型変数のアドレスを渡すと、誤動作する場合があります。

-signed_char

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-signed_char
```

- 省略時解釈
  char 型を符号なしとして扱います。

［詳細説明］

- char 型を符号付きとして扱います。

［備考］

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

-unsigned_char

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-unsigned_char
```

- 省略時解釈
  char 型を符号なしとして扱います。

［詳細説明］

- char 型を符号なしとして扱います。

［備考］

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

-signed_bitfield

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-signed_bitfield
```

- 省略時解釈
  符号なし型として扱います。

［詳細説明］

- 符号付き型として扱います。

-unsigned_bitfield

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-unsigned_bitfield
```

- 省略時解釈
  符号なし型として扱います。

［詳細説明］

- 符号なし型として扱います。

-auto_enum

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-auto_enum
```

- 省略時解釈
  列挙型サイズを signed long 型として処理します。

［詳細説明］

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

表 2.6　列挙型のとりうる値と型の関係

列挙子		選択される型	
最小値	最大値	-unsigned_char 選択時	-signed_char 選択時
-128	127	signed char	char
0	255	char	unsigned char
-32768	32767	signed short	signed short
0	65535	unsigned short	unsigned short
上記以外		int *1	int *1

【注意】
*1) -int_to_short 選択時は、符号付き 4 バイト整数型となります。

-bit_order

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  bit_order=right です。

［詳細説明］

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

-pack

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-pack
```

- 省略時解釈
  unpack です。すなわち、構造体メンバ、クラスメンバはそれぞれの型に応じたアライメント数を持ちます。また、構造体、クラスのアライメント数は、メンバの最大のアライメント数と同じになります。

［詳細説明］

- 構造体メンバ、クラスメンバのアライメント数を 1 にします。
- 構造体メンバのアライメント数は、#pragma pack 指令でも 1 にできます。オプションと #pragma の両方が指定された場合には、#pragma 指令を優先します。

［備考］

- 本オプション指定時の構造体メンバのアライメント数を以下に示します。

表 2.7 **pack** オプション指定時の構造体メンバ、クラスメンバのアライメント数

メンバの型	pack 指定あり	pack 指定なし
char/signed char/unsigned char/bool 注 1	1	1
signed short/unsigned short	1	2
signed int 注 2/unsigned int 注 2/signed long/unsigned long/ signed long long/unsigned long long/float/double/ long double/ ポインタ型	1	4

注 1. -lang=c オプションを指定した場合は、unsigned long と同じになります。

注 2. int_to_short オプションを指定した場合は、signed short/unsigned short と同じになります。

-unpack

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-unpack
```

- 省略時解釈
  unpack です。すなわち、構造体メンバ、クラスメンバはそれぞれの型に応じたアライメント数を持ちます。また、構造体、クラスのアライメント数は、メンバの最大のアライメント数と同じになります。

［詳細説明］

- 構造体メンバ、クラスメンバはそれぞれの型に応じたアライメント数を持ちます。
- 構造体、クラスのアライメント数は、メンバの最大のアライメント数と同じになります。

［備考］

- 本オプション指定時の構造体メンバのアライメント数を以下に示します。

表 2.8 **unpack** オプション指定時の構造体メンバ、クラスメンバのアライメント数

メンバの型	unpack 指定あり	unpack 指定なし
char/signed char/unsigned char/bool 注 1	1	1
signed short/unsigned short	2	2
signed int 注 2/unsigned int 注 2/signed long/unsigned long/ signed long long/unsigned long long/float/double/ long double/ ポインタ型	4	4

注 1. -lang=c オプションを指定した場合は、unsigned long と同じになります。

注 2. int_to_short オプションを指定した場合は、signed short/unsigned short と同じになります。

-exception

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-exception
```

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

［詳細説明］

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

［備考］

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

-noexception

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-noexception
```

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

［詳細説明］

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

［備考］

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

-rtti

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  rtti=off です。

［詳細説明］

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

［備考］

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

-fint_register

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  fint_register=0 です。

［詳細説明］

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

表 2.9 オプションとレジスタの関係

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

［備考］

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

-branch

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  branch=24 です。

［詳細説明］

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

-base

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

［詳細説明］

- プログラム全体で、ベースアドレスとして固定で使用する汎用レジスタを指定します。
- base=rom=< レジスタ A> を指定した場合は、const 変数のアクセスを指定したレジスタ A 相対で行うことを試みます。ただし、定数領域セクションのうち、最も 0 に近いアドレスと、最も 0xFFFFFFFF に近いアドレスとの差が 64KB ～ 256KB(*1) 以内でなければなりません。
  定数領域セクションは、次のセクション (セクション名変更前) が対象となります。
  C_1, C_2, C, C_8, C$VECT, C$INIT, C$VTBL, W, W_1, W_2, L
- base=ram=< レジスタ B> を指定した場合は、初期化変数および未初期化変数のアクセスは指定したレジスタ B 相対で行います。ただし、RAM データ領域セクションの、最も 0 に近いアドレスと最も 0xFFFFFFFF に近いアドレスとの差が 64KB ～ 256KB(*1) 以内でなければなりません。
  RAM データ領域セクションは、次のセクション (セクション名変更前) が対象となります。
  D_1, D_2, D, D_8, B_1, B_2, B, B_8
- < アドレス値 >=< レジスタ C> を指定した場合は、コンパイル時に割り付けアドレスが確定している領域のうち、アドレス値から 64KB ～ 256KB*1 以内の領域のアクセスを、指定したレジスタ C 相対で行います。

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

［備考］

- 異なる領域に対して同じレジスタを指定することはできません。
- レジスタはそれぞれの領域で 1 個だけ指定可能です。fint_register オプションで指定したレジスタを本オプションで指定した場合、エラーとなります。
- pid オプション選択時は、base=rom=< レジスタ > を選択できません。選択すると、警告として W0551149 メッセージを表示して base=rom=< レジスタ > の選択を無効とします。

-patch

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

［詳細説明］

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

-pic

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-pic
```

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

［詳細説明］

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

［例］

- 関数呼び出し （ただし、branch=32 の場合）

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

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

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

- 関数アドレスの取得

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

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

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

［備考］

- C++ または EC++ コンパイル時は、pic オプションを選択できません。選択すると、警告として W0511171 メッセージを表示して pic の選択を無効とします。
- PIC である関数のアドレスを、静的初期化の初期化式に使用しないでください。エラー E0523026 となります。
- PIC アドレスを静的初期化に用いる例 :

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

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

-pid

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  定数領域セクション C_8, C, C_2 および C_1、リテラルセクション L と switch 文分岐テーブルセクション W, W_2 および W_1 を PID（位置独立データ）としません。

［詳細説明］

- 定数領域セクション C_8, C, C_2 および C_1、リテラルセクション L と switch 文分岐テーブルセクション W, W_2 および W_1 を PID（位置独立データ）とします。
- PID のアクセスは、すべて PID レジスタからの相対アドレスで行います。これにより、PID はリンク後に任意のアドレスに配置することができます。
- PID 機能を実現するためには、汎用レジスタを 1 本消費します。
- ＜ PID レジスタ＞
- 下記の表の規則に基づき、fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお、fint_register の指定がない場合は R13 を選択します。

表 2.10 fint_register オプションと PID レジスタの関係

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

- PID レジスタは、PID のアクセスに使用する用途以外には使用されません。
- ＜パラメータ＞
- パラメータは、PID レジスタから定数領域セクションをアクセスする際の、オフセットの最大幅のビット数を 16 または 32 で選択します。
- オフセット幅を省略して pid オプションを選択した場合の解釈は、pid=16 です。pid=16 では、PID レジスタがアクセスできる定数領域セクションのサイズが 64KB ～ 256KB（アクセス幅により変動する）に制限されます。pid=32 では、PID レジスタがアクセスできる定数領域セクションのサイズに制限はありませんが、PID をアクセスするコードのサイズが増大します。
- なお、pid=32 と 外部シンボル割り付け情報が有効な map オプションを同時指定した場合は、割り付け情報により、PID レジスタによるアクセスが可能な場合は pid=16 と同じコードを生成します。

［例］

- const 修飾した外部参照シンボルをアクセス

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

- const 修飾した外部定義シンボルのアドレスを取得

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

［備考］

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

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

- PID 機能を利用するアプリケーションプログラムのスタートアップを作成する際は、「スタートアップ」ではなく、「アプリケーションのスタートアップ」を参照ください。
- pid オプション選択時は、同一の外部変数は、必ずファイル間で const 修飾を統一してください。これは、pid オプションは const 修飾のある変数を PID にするためです。もし const 修飾の統一が不明な外部変数がある場合は、pid オプション（PID 機能）は使用しないでください。
- pid オプション選択時に、ファイル指定のある map オプションを有効にした場合、ファイル間で同じ外部変数に対して const 修飾が統一されていない外部参照変数に対して警告 W0530809 を表示することがあります。表示された変数は PID として扱います。
- C++ または EC++ コンパイル時は、pid オプションを選択できません。選択すると、警告 W0511171 を表示して pid の選択を無効とします。
- pid オプション選択時は、base=rom=< レジスタ > を選択できません。選択すると、警告 W0551149 を表示して base=rom=< レジスタ > の選択を無効とします。
- pid オプションで選択された PID レジスタが、base オプションでも選択された場合はエラー E0511150 になります。
- pid オプションと nouse_pid_register オプションを同時に選択するとエラー E0511150 になります。
- アプリケーションおよび PID 機能の内容については、「PIC/PID 機能の利用」を参照してください。

-nouse_pid_register

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-nouse_pid_register
```

［詳細説明］

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

［備考］

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

-save_acc

＜コンパイル・オプション / マイコンオプション＞

［指定形式］

```
-save_acc
```

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

［詳細説明］

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

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

［備考］

- 生成される退避・回復コードは、#pragma interrupt に割り込み仕様 acc を選択したときに生成されるコードと同じものです。実際の退避・回復コードは、「4.2.3 #pragma 指令」の #pragma interrupt の項目を参照ください。
- 本オプション指定時は、割り込み時でもアキュムレータの値が保持されるため、C/C++ の演算式に対して MACLO 命令などの DSP 機能命令を用いたコードが生成されることがあります。

アセンブル・リンクオプション

＜コンパイル・オプション / アセンブル・リンクオプション＞
アセンブル、リンクオプションには、次のものがあります。

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

-asmcmd

＜コンパイル・オプション / アセンブル・リンクオプション＞

［指定形式］

```
-asmcmd=< ファイル名 >
```

［詳細説明］

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

［例］

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

と記述した場合、以下の 2 行のコマンド記述と同じ意味になります。

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

［備考］

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

-lnkcmd

＜コンパイル・オプション / アセンブル・リンクオプション＞

［指定形式］

```
-lnkcmd=< ファイル名 >
```

［詳細説明］

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

［例］

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

と記述した場合、以下の 3 行のコマンド記述と同じ意味になります。

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

［備考］

- 本オプションを複数回指定した場合、指定したすべてのサブコマンドファイルが有効となります。
- -lnkcmd オプションに渡すサブコマンドファイルの内容は、最適化リンケージエディタの -subcommand オプションの項目を参照ください。

-asmopt

＜コンパイル・オプション / アセンブル・リンクオプション＞

［指定形式］

```
-asmopt=["] ＜アセンブラオプション＞ ["]
```

［詳細説明］

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

［例］

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

と記述した場合、以下の 2 行のコマンド記述と同じ意味になります。

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

［備考］

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

-lnkopt

＜コンパイル・オプション / アセンブル・リンクオプション＞

［指定形式］

```
-lnkopt=["] ＜リンクオプション＞ ["]
```

［詳細説明］

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

［例］

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

と記述した場合、以下の 3 行のコマンド記述と同じ意味になります。

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

［備考］

- 本オプションを複数回指定した場合、指定したすべてのリンクオプションが有効となります。
- ひとつの -lnkopt で渡すことのできるリンクオプションはひとつだけです。複数のリンクオプションを渡す場合は、-lnkopt をその数だけ指定してください。

その他オプション

＜コンパイル・オプション / その他オプション＞
その他オプションには、次のものがあります。

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

-logo

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-logo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

- コピーライト表示が出力されます。

-nologo

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-nologo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

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

-euc

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-euc
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を EUC コードで扱います。

-sjis

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-sjis
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

-latin1

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-latin1
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を ISO-Latin1 コードで扱います。

-utf8

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-utf8
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を UTF-8 コードで扱います。

-big5

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-big5
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を Big5 コードで扱います。

［備考］

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

-gb2312

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-gb2312
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

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

［備考］

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

-outcode

＜コンパイル・オプション / その他オプション＞

［指定形式］

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

- 省略時解釈
  outcode=sjis です。

［詳細説明］

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

表 2.11 オプションと文字コードの関係（**outcode**）

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

［備考］

- outcode=big5 または outcode=gb2312 を指定する場合は、それぞれ big5 または gb2312 オプションもあわせて指定しなければなりません。

-subcommand

＜コンパイル・オプション / その他オプション＞

［指定形式］

```
-subcommand=< サブコマンドファイル名 >
```

［詳細説明］

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

［備考］

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

2.5.2 アセンブル・オプション

分類	オプション	説明
ソースオプション	-include	インクルード・ファイルの取り込み先フォルダを指定します。
	-define	マクロ定義を指定します。
	-chkpm	特権命令の記述の有無をチェックします。
	-chkfpu	単精度浮動小数点処理命令の記述の有無をチェックします。
	-chkdsp	DSP 機能命令の記述の有無をチェックします。
	-chkdpfpu 【V3.01.00 以降】	倍精度浮動小数点処理命令の記述の有無をチェックします。
オブジェクトオプション	-output	リロケータブル・ファイル名を指定します。
	-debug	オブジェクト・ファイルにデバッグ情報を出力します。
	-nodebug	オブジェクト・ファイルにデバッグ情報を出力しません。
	-goptimize	モジュール間最適化用付加情報を出力します。
	-fpu	単精度浮動小数点処理命令の記述が可能になります。
	-nofpu	単精度浮動小数点処理命令の記述をエラーにします。
	-dpfpu 【V3.01.00 以降】	倍精度浮動小数点処理命令の記述が可能になります。
	-nodpfpu 【V3.01.00 以降】	倍精度浮動小数点処理命令の記述をエラーにします。
	-bank 【V3.01.00 以降】	レジスター括退避機能用の命令の記述が可能になります。
	-nobank 【V3.01.00 以降】	レジスター括退避機能用の命令の記述をエラーにします。
	-create_unfilled_area 【V2.03.00 以降】	.OFFSET が作る空き領域に対してデータを出力しません。
リストオプション	-listfile	アセンブル・リスト・ファイル出力します。
	-nolistfile	アセンブル・リスト・ファイル出力しません。
	-show	ソース・リスト・ファイルの内容を指定します。
マイコンオプション	-isa	命令セット・アーキテクチャを選択します。
	-cpu	マイコン種別を選択します。
	-endian	エンディアン選択します。
	-fint_register	高速割り込み関数でのみ使用する汎用レジスタを選択します。
	-base	ROM,RAM 用ベースレジスタ選択します。
	-patch	CPU タイプ特有の問題を回避するかどうかを選択します。
	-pic	PIC 機能を有効にします。
	-pid	PID 機能を有効にします。
	-nouse_pid_register	PID レジスタをコード生成に使用しません。

分類	オプション	説明
その他オプション	-logo	コピーライトを出力します。
	-nologo	コピーライトを出力しません。
	-subcommand	コマンドオプションを取り込むファイルを指定します。
	-euc	入力プログラムの文字コードを EUC コードと解釈します。
	-sjis	入力プログラムの文字コードを SJIS コードと解釈します。
	-latin1	入力プログラムの文字コードを ISO-Latin1 コードと解釈します。
	-big5	入力プログラムの文字コードを BIG5 コードと解釈します。
	-gb2312	入力プログラムの文字コードを GB2312 コードと解釈します。
	-utf8 【V2.04.00 以降】	入力プログラムの文字コードを UTF-8 コードと解釈します。

ソースオプション

＜アセンブル・オプション / ソースオプション＞
ソースオプションには、次のものがあります。

- -include
- -define
- -chkpm
- -chkfpu
- -chkdsp
- -chkdpfpu 【V3.01.00 以降】

-include

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-include = < パス名 >[,…]
```

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

［詳細説明］

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

［例］

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

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

-define

＜アセンブル・オプション / ソースオプション＞

［指定形式］

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

［詳細説明］

- マクロ名を対応する文字列に置き換えます。
（ソースファイル先頭で .DEFINE 指示命令を記述するのと同じです）

［備考］

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

-chkpm

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkpm
```

［詳細説明］

- 特権命令を記述すると警告 W0551011 を出力します。

［備考］

- 特権命令の詳細な説明については、RX のソフトウェアマニュアルを参照してください。

-chkfpu

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkfpu
```

［詳細説明］

- 単精度浮動小数点処理命令を記述すると警告 W0551012 を出力します。

［備考］

- 単精度浮動小数点処理命令の詳細な説明については、RX のソフトウェアマニュアルを参照してください。

-chkdsp

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkdsp
```

［詳細説明］

- DSP 機能命令を記述すると警告 W0551013 を出力します。

［備考］

- DSP 機能命令の詳細な説明については、RX のソフトウェアマニュアルを参照してください。

-chkdpfpu 【V3.01.00 以降】

＜アセンブル・オプション / ソースオプション＞

［指定形式］

```
-chkdpfpu
```

［詳細説明］

- 倍精度浮動小数点処理命令の記述に対して警告 W0551017 を出力します。

［備考］

- 倍精度浮動小数点処理命令の詳細な説明については、RX のソフトウェアマニュアルを参照してください。

オブジェクトオプション

＜アセンブル・オプション / オブジェクトオプション＞
オブジェクトオプションには、次のものがあります。

- -output
- -debug
- -nodebug
- -goptimize
- -fpu
- -nofpu
- -dpfpu 【V3.01.00 以降】
- -nodpfpu 【V3.01.00 以降】
- -bank 【V3.01.00 以降】
- -nobank 【V3.01.00 以降】
- -create_unfilled_area【V2.03.00 以降】

-output

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-output = < 出力ファイル名 >
```

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

［詳細説明］

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

-debug

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-debug
```

- 省略時解釈
  リロケータブルファイル内にデバッグ情報を出力しません。

［詳細説明］

- リロケータブルファイル内にデバッグ情報を出力します。

-nodebug

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-nodebug
```

- 省略時解釈
  リロケータブルファイル内にデバッグ情報を出力しません。

［詳細説明］

- リロケータブルファイル内にデバッグ情報を出力しません。

-goptimize

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-goptimize
```

- 省略時解釈
  モジュール間最適化用付加情報を出力しません。

［詳細説明］

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

-fpu

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-fpu
```

- 省略時解釈
  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。
  CPU に RX200 を選択（*2）した場合は nofpu、それ以外は fpu です。

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

［詳細説明］

- 単精度浮動小数点処理命令の記述が可能になります。

［備考］

- CPU に RX200 を選択した場合、fpu を指定するとエラーになります。
- 単精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。

-nofpu

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-nofpu
```

- 省略時解釈
  ISA（*1）で命令セットアーキテクチャを選択した場合は fpu です。
  CPU に RX200 を選択（*1）した場合は nofpu、それ以外は fpu です。

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

［詳細説明］

- 単精度浮動小数点処理命令の記述をエラーにします。

［備考］

- 単精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。
- 本オプションを指定した場合、単精度浮動小数点処理命令と、制御レジスタ FPSW の記述がエラーになります。

-dpfpu 【V3.01.00 以降】

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-dpfpu
```

- 省略時解釈
  nodpfpu です。

［詳細説明］

- 倍精度浮動小数点処理命令の記述が可能になります。

［備考］

- CPU（*1）が選択された場合及び ISA（*2）として RXv1 または RXv2 が選択された場合、dpfpu を指定するとエラーになります。
- 倍精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。
- nofpu が指定されている場合、dpfpu を指定するとエラーになります。

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

-nodpfpu 【V3.01.00 以降】

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-nodpfpu
```

- 省略時解釈
  nodpfpu です。

［詳細説明］

- 倍精度浮動小数点処理命令の記述をエラーにします。

［備考］

- 倍精度浮動小数点処理命令の具体的な内容については、RX のソフトウェアマニュアルを参照してください。
- 本オプションを指定した場合、倍精度浮動小数点処理命令、倍精度浮動小数点データレジスタ（DR0 ～ DR15）及び倍精度浮動小数点制御レジスタ（DPSW、DMCR、DECNT、DEPC）の記述がエラーとなります。

-bank 【V3.01.00 以降】

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-bank
```

- 省略時解釈
  CPU（*1）が選択された場合及び ISA（*2）として RXv1 または RXv2 が選択された場合は -nobank です。
  それ以外の場合は -bank です。

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

［詳細説明］

- レジスター括退避機能用の命令の記述が可能になります。

-nobank 【V3.01.00 以降】

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-nobank
```

- 省略時解釈
  CPU（*1）が選択された場合及び ISA（*2）として RXv1 または RXv2 が選択された場合は -nobank です。
  それ以外の場合は -bank です。

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

［詳細説明］

- レジスター括退避機能用の命令をエラーにします。

-create_unfilled_area【V2.03.00 以降】

＜アセンブル・オプション / オブジェクトオプション＞

［指定形式］

```
-create_unfilled_area
```

［詳細説明］

- S レコード・ファイル (～ .mot) またはヘキサ・ファイル (～ .hex) を出力する際に、アセンブリ言語において .OFFSET 疑似命令を記述した箇所に作られる空き領域に対し、データを出力しないようにします。
- 本オプションの機能を利用するには、rlink で S レコード・ファイルおよびヘキサ・ファイルを作成するときだけではなく、ccrx コマンドまたは asrx コマンドでオブジェクト・ファイル (～ .obj) を作成するときにも指定する必要があります。

［備考］

- 本オプションを使用すると、.OFFSET 疑似命令ごとに、次のような形式のシンボル (*1) を追加します。
  ___$_<FileName>_<SectionName>_<IDNumber>s__unfilled_area
  ___$_<FileName>_<SectionName>_<IDNumber>e__unfilled_area
  ここで、<FileName> にはソースファイル名、<SectionName> にはセクション名、<IDNumber> には 1 から始まる数値が入ります。

注 1. これらの形式のシンボルは予約されており、お客様のソースコードに直接記述することはできません。

リストオプション

＜アセンブル・オプション / リストオプション＞
リストオプションには、次のものがあります。

- -listfile
- -nolistfile
- -show

-listfile

＜アセンブル・オプション / リストオプション＞

［指定形式］

```
-listfile[=< ファイル名 >]
```

- 省略時解釈
  アセンブルリストファイルは出力しません。

［詳細説明］

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

-nolistfile

＜アセンブル・オプション / リストオプション＞

［指定形式］

```
-nolistfile
```

- 省略時解釈
  アセンブルリストファイルは出力しません。

［詳細説明］

- アセンブルリストファイルは出力しません。

-show

＜アセンブル・オプション / リストオプション＞

［指定形式］

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

［詳細説明］

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

表 2.12 **show** オプション指定一覧

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

マイコンオプション

＜アセンブル・オプション / マイコンオプション＞
マイコンオプションには、次のものがあります。

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

-isa

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

```
-isa = { rxv1 | rxv2 | rxv3 }
```

- 省略時解釈
  環境変数 ISA_RX の内容に従います。

［詳細説明］

- オブジェクトファイルを生成するのに用いる命令セットアーキテクチャを指定します。

［備考］

- 選択した命令セットアーキテクチャでサポートされない命令の記述はエラーとなります。
- -nofpu と -fpu のいずれも選択されていない場合に本オプションを指定すると、-fpu が自動的に選択されます。
- -cpu、環境変数 CPU_RX 及び環境変数 ISA_RX のいずれも設定されていない場合に本オプションを省略すると、エラーになります。
- -cpu と本オプションを同時に指定するとエラーになります。

-cpu

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  環境変数 CPU_RX に従います。

［詳細説明］

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

［備考］

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

【注意】

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

-endian

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  -endian=little です。

［詳細説明］

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

-fint_register

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

- 省略時解釈
  fint_register=0 です。

［詳細説明］

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

［備考］

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

-base

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

［詳細説明］

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

［備考］

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

-patch

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

［詳細説明］

- CPU の品種ごとに特有の問題を回避します。
- -patch=rx610 を指定した場合、RX610 グループで問題となる MVTIPL 命令を未定義命令として扱います。MVTIPL は命令と認識されずにエラーメッセージ E0552113 が出力されます。

-pic

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

```
-pic
```

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

［詳細説明］

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

［備考］

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

-pid

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

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

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

［詳細説明］

- PID 機能が有効な状態でコード生成されたことを表すリロケータブルオブジェクトを生成します。
- ＜PID レジスタ＞
  下記の表の規則に基づき、fint_register オプションの指定に応じて R9 から R13 のうちの 1 本を選択します。なお、fint_register の指定がない場合は R13 を選択します。

表 2.13 **fint_register** オプションと **PID** レジスタの関係

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

- PID レジスタは、PID のアクセスに使用する用途以外には使用されません。
- ＜パラメータ＞
  パラメータの意味は、コンパイラの同名オプションと同じです。

［備考］

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

-nouse_pid_register

＜アセンブル・オプション / マイコンオプション＞

［指定形式］

```
-nouse_pid_register
```

［詳細説明］

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

［備考］

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

その他オプション

＜アセンブル・オプション / その他オプション＞
その他オプションには、次のものがあります。

- -logo
- -nologo
- -subcommand
- -euc
- -sjis
- -latin1
- -big5
- -gb2312
- -utf8 【V2.04.00 以降】

-logo

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-logo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

- コピーライト表示が出力されます。

-nologo

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-nologo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

- コピーライトの表示の出力が抑止されます。

-subcommand

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-subcommand=< ファイル名 >
```

［詳細説明］

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

［例］

- ＜サブコマンドファイル opt.sub の内容＞

```
-listfile
-debug
```

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

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

-euc

- <アセンブル・オプション / その他オプション>

[指定形式]

```
-euc
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

[詳細説明]

- 文字列、文字定数およびコメント内の文字を EUC コードで扱います。

-sjis

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-sjis
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

-latin1

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-latin1
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を ISO-Latin1 コードで扱います。

-big5

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-big5
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を Big5 コードで扱います。

-gb2312

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-gb2312
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

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

-utf8 【V2.04.00 以降】

＜アセンブル・オプション / その他オプション＞

［指定形式］

```
-utf8
```

- 省略時解釈
  文字列、文字定数およびコメント内の文字を SJIS コードで扱います。

［詳細説明］

- 文字列、文字定数およびコメント内の文字を UTF-8 コードで扱います。

2.5.3 最適化リンケージエディタ（rlink）・オプション

分類	オプション	説明
入力オプション	-Input	リロケータブル・ファイルを指定します。
	-library	ライブラリ・ファイルを指定します。
	-binary	バイナリ・ファイルを指定します。
	-define	シンボル定義を指定します。
	-entry	エントリシンボル / アドレスを指定します。
	-noprelink	プレリンカを起動しません。
	-allow_duplicate_module_name 【V3.02.00 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
出力オプション	-form	出力ファイル形式を選択します。
	-debug	デバッグ情報をロード・モジュール・ファイル内に出力します。
	-sdebug	デバッグ情報を .dbg ファイルに出力します。
	-nodebug	デバッグ情報を出力しません。
	-record	レコードサイズを選択します。
	-end_record 【V2.07.00 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	出力ファイル名を指定します。
	-map	外部シンボル割り付け情報ファイルを出力します。
	-space	出力範囲のメモリの空き領域をデータで充填します。
	-message	インフォメーションレベル・メッセージを出力します。
	-nomessage	出力抑止メッセージを指定します。
	-msg_unused	参照されない外部定義シンボルをメッセージ出力します。
	-byte_count	データ・レコードのバイト数を指定します。
	-fix_record_length_and_align 【V2.08.00 以降】	データレコードの出力フォーマットを固定します。
	-crc	CRC コードの出力形式を指定します。
	-padding	終端にパディングを埋め込みます。
	-vectn	可変ベクタの特定ベクタ番号へのアドレスを指定します。（RX ファミリ、M16C ファミリ向け）
	-vect	可変ベクタの空き領域へのアドレスを指定します。（RX ファミリ、M16C ファミリ向け）
	-split_vect 【V3.00.00 以降】	ベクタテーブルセクションを分割して生成します。
	-jump_entries_for_pic	ジャンプ・テーブル・ファイルを出力します。（RX ファミリの PIC 機能向け）
	-cfi【Professional 版のみ】【V2.08.00 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 【V2.08.00 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関数シンボルまたはアドレスを追加します。
	-cfi_ignore_module【V2.08.00 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。
	-create_unfilled_area【V2.03.00 以降】	.OFFSET が作る空き領域に対してデータを出力しません。
リストオプション	-list	リンケージ・リスト・ファイルを出力します。
	-show	リンケージ・リスト・ファイルの出力する内容を選択します。

分類	オプション	説明
最適化オプション	-optimize	リンク時最適化内容を選択します。
	-nooptimize	リンク時最適化を行いません。
	-samesize	共通コード統合の対象となる最小サイズを指定します。
	-symbol_forbid	未参照シンボル削除抑止シンボルを指定します。
	-samecode_forbid	共通コード統合抑止シンボルを指定します。
	-section_forbid	最適化抑止セクションを指定します。
	-absolute_forbid	最適化抑止アドレス範囲を指定します。
	-ALLOW_OPTIMIZE_ENTRY_BLOCK【V3.06.00 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。
セクションオプション	-start	セクションの開始アドレスを指定します。
	-fsymbol	外部定義シンボルをファイル出力するセクションを指定します。
	-aligned_section	セクション・アライメントを 16 バイトにします。
ベリファイオプション	-cpu	アドレスの整合性をチェックします。
	-contiguous_section	セクション分割の対象外セクションを指定します。
その他オプション	-s9	S9 レコードを終端に出力します。
	-stack	スタック情報ファイルを出力します。
	-compress	デバッグ情報を圧縮します。
	-nocompress	デバッグ情報を圧縮しません。
	-memory	リンク時に使用するメモリ量を選択します。
	-rename	変更するシンボル名、セクション名を指定します。
	-lib_rename 【V3.01.00 以降】	ライブラリから入力されたシンボル名、セクション名を変更します。
	-delete	削除するシンボル名、モジュール名を指定します。
	-replace	置換するライブラリ・モジュール指定します。
	-extract	ライブラリファイルから抽出するモジュールを指定します。
	-strip	アブソリュートファイル、ライブラリファイルのデバッグ情報を削除します。
	-change_message	変更するインフォメーション、ウォーニング、エラーレベルのメッセージレベル、メッセージを指定します。
	-hide	ローカルシンボル名情報を削除します。
	-total_size	リンク後の合計セクションサイズを標準出力に表示します。
	-verbose 【V3.03.00 以降】	詳細情報を標準エラー出力に表示します。
サブコマンドファイルオプション	-subcommand	コマンドオプションを取り込むファイルを指定します。

分類	オプション	説明
残りのオプション	-logo	コピーライトを出力します。
	-nologo	コピーライトを出力しません。
	-end	END より前に指定したオプション列を実行します。
	-exit	オプション指定の終了を指定します。

入力オプション

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞
入力オプションには、次のものがあります。

- -Input
- -library
- -binary
- -define
- -entry
- -noprelink
- -allow_duplicate_module_name 【V3.02.00 以降】

-Input

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-library

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-library= < ファイル名 >[,…]
```

［詳細説明］

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

［例］

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

-binary

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

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

［備考］

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

-define

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-entry

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-noprelink

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-noprelink
```

- 省略時解釈
  プレリンカを起動します。

［詳細説明］

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

［備考］

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

-allow_duplicate_module_name 【V3.02.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 入力オプション＞

［指定形式］

```
-allow_duplicate_module_name
```

［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。

［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。

出力オプション

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞
出力オプションには、次のものがあります。

- -form
- -debug
- -sdebug
- -nodebug
- -record
- -end_record 【V2.07.00 以降】
- -rom
- -output
- -map
- -space
- -message
- -nomessage
- -msg_unused
- -byte_count
- -fix_record_length_and_align 【V2.08.00 以降】
- -crc
- -padding
- -vectn
- -vect
- -split_vect 【V3.00.00 以降】
- -jump_entries_for_pic
- -cfi 【Professional 版のみ】【V2.08.00 以降】
- -cfi_add_func 【V2.08.00 以降】
- -cfi_ignore_module 【V2.08.00 以降】
- -create_unfilled_area【V2.03.00 以降】

-form

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

- 省略時解釈
  form=absolute です。

［詳細説明］

- 出力形式を指定します。
- サブオプションの一覧を表 2.14 に示します。

表 2.14 form オプションのサブオプション一覧

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

［備考］

出力形式と入力ファイル、他オプションとの関係を表 2.15 に示します。

表 2.15 出力形式と入力ファイル、他オプションとの関係

No	出力形式	指定オプション	入力可能なファイル形式	指定可能なオプション[注 1]
1	Absolute	strip あり	アブソリュートファイル	input, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, binary, debug/nodebug, sdebug, cpu, start, rom, entry, output, map, hide, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, section_forbid, absolute_forbid, compress, rename, lib_rename, delete, define, fsymbol, stack, noprelink, memory, msg_unused, show={symbol,reference,xreference,total_size, vector,relocation_attribute}, jump_entries_for_pic, aligned_section, padding, vectn, vect, split_vect
2	Relocate	extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, debug/nodebug, output, hide, rename, lib_rename, delete, noprelink, msg_unused, show=symbol,xreference,total_size
3	Object	extract あり	ライブラリファイル	library, output
4	Hexadecimal Stype Binary		オブジェクトファイル リロケータブルファイル バイナリファイル ライブラリファイル	input, library, binary, cpu, start, rom, entry, output, map, space, optimize/nooptimize, samesize, symbol_forbid, samecode_forbid, section_forbid, absolute_forbid, rename, lib_rename, delete, define, fsymbol, stack, noprelink, record, end_record, s9[注 2], byte_count[注 3], fix_record_length_and_align, memory, msg_unused, show=symbol,reference,xreference,total_size, vector, jump_entries_for_pic, aligned_section, padding, vectn, vect, split_vect
			アブソリュートファイル	input, output, record, end_record, s9[注 2], byte_count[注 3], fix_record_length_and_align, show=symbol, reference, xreference
5	Library	strip あり	ライブラリファイル	library, output, memory[注 4], show=symbol, section
		extract あり	ライブラリファイル	library, output
		上記以外	オブジェクトファイル リロケータブルファイル	input, library, output, hide, rename, delete, replace, noprelink, memory[注 4], show=symbol, section, allow_duplicate_module_name

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

注 2. s9 は出力形式が form=stype のときだけ指定できます。

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

注 4. hide 指定する場合は使用できません。

-debug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-debug
```

- 省略時解釈
  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

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

［備考］

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

-sdebug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-sdebug
```

- 省略時解釈
  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

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

［備考］

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

-nodebug

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-nodebug
```

- 省略時解釈
  出力ファイル中にデバッグ情報を出力します。

［詳細説明］

- デバッグ情報を出力しません。

［備考］

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

-record

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

- 省略時解釈
  それぞれのアドレスに合わせて混在したデータレコードを出力します。

［詳細説明］

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

［備考］

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

-end_record 【V2.07.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-end_record = { S7 | S8 | S9 }
```

［詳細説明］

- モトローラ S 形式ファイルのエンドレコードを指定します。
- 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。
- 本オプション省略時は、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。

［例］

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

- 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。

［備考］

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

-rom

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

［詳細説明］

- 初期化データ領域の ROM 用、RAM 用領域を確保し、ROM セクション内定義シンボルを RAM セクション内アドレスになるようリロケーションします。
- ROM セクションには初期値のあるリロケータブルセクションを指定します。
- RAM セクションには存在しないセクションまたはサイズ 0 のリロケータブルセクションを指定します。
- ROM セクション名、RAM セクション名には、ワイルドカード記号 '*' を使用することができます。【V3.06.00 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、ROM セクション名のワイルドカード表現と一致する場合、RAM セクション名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 D、D_1、D_2 の 3 つあるとき、-rom=D*=R* と指定すると、R、R_1、R_2 の 3 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=D*=R*        # 問題なし
-rom=D*=*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。

［例］

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

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

［備考］

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

-output

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

- 省略時解釈
  < 先頭入力ファイル名 >.< デフォルト拡張子 > です。
- デフォルト拡張子は、次のようになります。
  form=absolute :「abs」、form=relocate :「rel」、form=object :「obj」、form=library :「lib」、form=hexadecimal :「hex」、form=stype :「mot」、form=binary :「bin」

［詳細説明］

- 出力ファイル名を指定します。form={absolute | hexadecimal | stype | binary} のときは、複数ファイルを指定できます。アドレスは 16 進数で指定します。先頭が A ～ F の場合は先にセクションを検索し、該当するセクションがなければアドレスと判断します。先頭に 0 を付加した場合は常にアドレスと解釈します。
- 【V3.00.00 以降】ロードアドレスを指定すると、インテル拡張ヘキサファイルまたはモトローラ S 形式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指定した値に変更します。

［例］

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

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

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

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

［備考］

- マイコン種別が RX ファミリでビッグエンディアンのときに、セクション単位で出力する場合は、セクションサイズを 4 の倍数にしてください。
- 【V3.00.00 以降】ロードアドレスは form=hexadecimal または form=stype の場合に指定できます。

-map

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-map [= < ファイル名 >]
```

［詳細説明］

- コンパイラが外部変数アクセス最適化で使用する外部シンボル割り付け情報ファイルを出力します。
- < ファイル名 > を指定しなかった場合は、output オプションで指定したファイル名、もしくは先頭入力ファイル名で、拡張子が bls のファイルを出力します。
- 外部シンボル割り付け情報ファイル作成時の変数宣言順と、再コンパイル後のオブジェクトを読み込んだ時の変数宣言順が変わっている場合はエラーを出力します。
- 以下に該当する場合、リンカは外部シンボル割り付け情報ファイルと、-list オプション指定がある場合はリンケージ・リスト・ファイルを出力して正常終了します。このときロード・モジュール・ファイルは出力しません。
    - 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

  外部シンボル割り付け情報ファイルには、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出力します。【V2.06 以降】

［備考］

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

-space

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-space [ = {< 数値 > | Random}]
```

［詳細説明］

- 出力範囲のメモリの空き領域を、ユーザが指定するデータで充填します。
- 充填するデータとしては、乱数、もしくは 16 進数の数値を指定することができます。
- 空きエリアを埋める方法は、出力範囲指定方法によって下記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
    - 指定されたセクション間に空きが存在した場合に指定データを出力
  - output オプションの出力範囲がアドレス範囲指定の場合
    - 指定された範囲内に空きが存在した場合に指定データを出力
  - 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領域が存在する場合に指定データを出力
    - 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

［備考］

- 本オプションにてサブオプションの指定がされなかった場合は、空きエリアへの出力は行いません。
- 本オプションは form={binary|stype|hexadecimal} オプションを指定した場合か、または -fix_record_length_and_align オプションを指定した場合にのみ有効となります。

-message

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-message
```

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

［詳細説明］

- インフォメーションレベルメッセージを出力します。

-nomessage

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

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

［詳細説明］

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

［例］

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

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

-msg_unused

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-msg_unused
```

［詳細説明］

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

［例］

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

［備考］

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

-byte_count

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-byte_count=< 数値 >
```

- 省略時解釈
  インテル HEX 形式ファイルを生成する場合、FF（16 進数）を指定したものとみなします。
  モトローラ S 形式ファイルを生成する場合、10（16 進数）を指定したものとみなします。

［詳細説明］

- インテル HEX 形式ファイルまたはモトローラ S 形式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 HEX 形式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 形式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）

［例］

データレコード長として 16byte（16 進数で 10）を指定します。

```
-byte_count=10
```

［備考］

- 生成するファイル形式がインテル HEX 形式ファイル（form=hex）またはモトローラ S 形式ファイル（form=stype）ではない場合、本オプションは無効です。

-fix_record_length_and_align 【V2.08.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-fix_record_length_and_align=< アライメント数 >
```

［詳細説明］

- インテル HEX 形式ファイルまたはモトローラ S 形式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により、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。
- 空き領域には、space オプションが指定された場合はその値、space オプションが指定されていない場合は crc オプションがないとき 0 を出力します。crc オプションがあるとき FF を出力します。

［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。

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

［備考］

- 生成するファイル形式がインテル HEX 形式ファイル（form=hex）でもモトローラ S 形式ファイル（form=stype）でもない場合、本オプションは無効です。

-crc

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-crc = ＜サブオプション＞
       ＜サブオプション＞: ＜出力位置＞=＜計算範囲＞[/＜演算方法＞][(＜初期値＞)][:＜エンディアン＞]
                          ＜出力位置＞: ＜アドレス＞
                          ＜計算範囲＞: { ＜先頭アドレス＞-＜終了アドレス＞ | ＜セクション＞ }[,...]
                          ＜演算方法＞: { CCITT | 16-CCITT-MSB | 16-CCITT-MSB-LITTLE-4 |
                                          16-CCITT-MSB-LITTLE-2 | 16-CCITT-LSB | 16 |
                                          SENT-MSB | 32-ETHERNET }
                          ＜初期値＞  : ＜初期値＞
                          ＜エンディアン＞: { BIG | LITTLE }[-＜サイズ＞-＜オフセット＞]
```

［詳細説明］

- 指定した範囲のセクションのデータを、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法で出力します。
- 演算方法には以下のいずれかを指定することができます。演算方法の指定を省略した場合は、CCITT を指定したものとして演算を行います。

表 2.16　演算方法一覧

演算方法	内容
CCITT	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 【V2.04.00 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 【V2.04.00 以降】	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 【V2.04.00 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 【V2.04.00 以降】	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	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 【V2.04.00 以降】	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 【V2.04.00 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。

- ＜初期値＞として指定可能な値の範囲は、演算方法が 32-ETHERNET では 0x0 ～ 0xFFFFFFFF、SENT-MSB では 0x0 ～ 0xF、それ以外では 0x0 ～ 0xFFFF です。
- ＜初期値＞を省略した場合は、演算方法が SENT-MSB では 0x5、CCITT では 0xFFFF、32-ETHERNET では 0xFFFFFFFF、それ以外は 0x0 を指定したものとして演算を行います。

- 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が省略された場合、サイズは 2 バイトで、オフセットは 0 とします。
- 計算範囲にある空き領域は space オプションが指定されていない場合は、space=FF が指定されていると仮定して、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、生成コードに 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。【V3.05.00 以降】

［例］

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

- crc オプション：-crc=2FFE=1000-2FFD
  - 0x1000 ～ 0x2FFD の領域に対して CRC 演算を行い、その結果を 0x2FFE 番地に出力します。
  - 計算範囲にある空き領域は space オプションが指定されていない場合、space=0xFF が指定されていると仮定して、CRC 演算を行います。
- output オプション：-output=out.mot=1000-2FFF
  - space オプションが指定されていないため、空きの領域は「out.mot」ファイルに出力されません。CRC 演算は、空き領域では 0xFF で計算を行いますが、0xFF を埋めることはありません。

    注 1. CRC 出力位置は、計算範囲に含むことは出来ません。

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

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

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

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

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

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

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

- 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］

- 複数のロード・モジュール・ファイル入力時、ワーニングを出力して本オプションを無視します。
- 出力形式が form={hexadecimal | stype | bin} の場合に有効です。これら以外の場合はエラーを出力して終了します。
- space オプションが指定されていない場合で、計算範囲に出力されない空き領域があるとき、空き領域には 0xFF が設定されているものとして CRC の計算が行われます。
- CRC 演算の計算範囲にオーバーレイ指定されている領域が含まれる場合はエラーを出力して終了します。
- エンディアンの指定で、サイズとオフセットには、以下が指定できます。これ以外の場合はエラーを出力して終了します。
    - LITTLE
    - LITTLE-2-0
    - LITTLE-4-0
    - BIG
    - BIG-2-0
    - BIG-4-0
- サンプルコード
  crc オプションで計算された CRC 演算結果を比較するためのサンプルコードです。
  サンプルコードのプログラムは、rlink の CRC 演算結果と一致します。
  演算方法 CRC-CCITT の場合（初期値 0xFFFF, MSB ファースト , 反転出力）

```
typedef  unsigned   char    uint8_t;
typedef  unsigned   short   uint16_t;
typedef  unsigned   long    uint32_t;
uint16_t CRC_CCITT(uint8_t *pData, uint32_t iSize)
{
    uint32_t  ui32_i;
    uint8_t       *pui8_Data;
    uint16_t  ui16_CRC = 0xFFFFu;
    pui8_Data = (uint8_t *)pData;
    for(ui32_i = 0; ui32_i < iSize; ui32_i++)
    {
        ui16_CRC  = (uint16_t)((ui16_CRC >> 8u) |
        ((uint16_t)((uint32_t)ui16_CRC << 8u)));
        ui16_CRC ^= pui8_Data[ui32_i];
        ui16_CRC ^= (uint16_t)((ui16_CRC & 0xFFu) >> 4u);
        ui16_CRC ^= (uint16_t) ((ui16_CRC << 8u) << 4u);
        ui16_CRC ^= (uint16_t)(((ui16_CRC & 0xFFu) << 4u) << 1u);
    }
    ui16_CRC = (uint16_t)( 0x0000FFFFul &
    ((uint32_t)~(uint32_t)ui16_CRC) );
    return ui16_CRC;
}
```

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

```
#define  POLYNOMIAL  0xa001      // 生成多項式 CRC-16
typedef  unsigned    char     uint8_t;
typedef  unsigned    short    uint16_t;
typedef  unsigned    long     uint32_t;
uint16_t CRC16(uint8_t *pData, uint32_t iSize)
{
    uint16_t crcdData = (uint16_t)0;
    uint32_t data = 0;
    uint32_t i,cycLoop;
    for(i=0;i<iSize;i++){
        data = (uint32_t)pData[i];
        crcdData = crcdData ^ data;
        for (cycLoop = 0; cycLoop < 8; cycLoop++) {
            if (crcdData & 1) {
                crcdData = (crcdData >> 1) ^ POLYNOMIAL;
            } else {
                crcdData = crcdData >> 1;
            }
        }
    }
    return crcdData;
}
```

-padding

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-padding
```

［詳細説明］

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

［例］

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

P セクションのアライメント :4 バイト
P セクションのサイズ :0x06 バイト
C セクションのアライメント :1 バイト
C セクションのサイズ :0x03 バイト
の場合、
P セクションに 2 バイトのパディングデータを埋め込んで、サイズを 0x08 バイトにしてリンクする。

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

P セクションのアライメント :4 バイト
P セクションのサイズ :0x06 バイト
C セクションのアライメント :1 バイト
C セクションのサイズ :0x03 バイト
の場合、
P セクションに 2 バイトのパディングデータを埋め込んで、サイズを 0x08 バイトにしてリンクすると、C セクションと重複してしまうため、E0562231 エラーを出力する。

［備考］

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

-vectn

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

［詳細説明］

- 可変ベクタテーブルセクションの特定ベクタ番号に対して、オプションで指定されたアドレスを設定します。
- 本オプションを使用した場合、ソース上に割り込み関数記述がなくても、可変ベクタテーブルセクションを作成し、テーブルへアドレスを設定します。
- ＜ベクタ番号＞は、10 進数で 0 ～ 255 の範囲で指定してください。
- ＜シンボル＞は、対象関数の外部名で指定してください。
- ＜アドレス＞は、指定アドレスを 16 進数で指定してください。
- 【V3.00.00 以降】split_vect の指定がないとき、vectn 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 の指定があるときは、vectn 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を生成しません。

［例］

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

［備考］

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

-vect

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-vect={< シンボル >|< アドレス >}
```

［詳細説明］

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

［備考］

ユーザが可変ベクタテーブルセクションをソースプログラムで作成している場合、可変ベクタテーブルの自動生成は行わないため、本オプションは無効になります。
{< シンボル >|< アドレス >} の記述で、先頭を 0 と記述したものはすべてアドレスとして判断します。

-split_vect 【V3.00.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-split_vect
```

［詳細説明］

- ベクタテーブルセクションをベクタテーブル番号別に分割して生成します。
- ベクタテーブル番号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。

［例］

- ベクタテーブル番号 14 に対してベクタテーブルセクション “C$VECT14” を生成します。

```
-vectn=14=__dummy -split_vect
```

［備考］

- 本オプションは、-vect オプション、-form={object | relocate | library} オプション、-strip オプションまたは -extract オプションを同時に指定した場合、無効になります。

-jump_entries_for_pic

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

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

［詳細説明］

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

［例］

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

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

- [test.jmp の出力例]

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

［備考］

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

-cfi 【Professional 版のみ】 【V2.08.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-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】【V2.08.00 以降】」を参照してください。
- リンカは関数リストを C 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 C 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、最適化リンケージエディタ (rlink)・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、最適化リンケージエディタ (rlink)・オプション -cfi_ignore_module を指定してください。

-cfi_add_func 【V2.08.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-cfi_add_func={ ＜関数シンボル＞ | ＜関数アドレス＞ }[,{ ＜関数シンボル＞ | ＜関数アドレス＞ }]...
```

［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、関数のシンボル又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】【V2.08.00 以降】」を参照してください。
- アドレスは 16 進数で指定します。
- 指定した関数シンボル又またはアドレスが（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又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無い場合はエラーとなります。

［例］

- C ソースの sub1 関数、関数アドレス 0x100、C ソースの sub2 関数を関数リストに登録します。

```
-cfi_add_func=_sub1,100 -cfi_add_func=_sub2
```

-cfi_ignore_module 【V2.08.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-cfi_ignore_module = ＜サブオプション＞[,...]
              ＜サブオプション＞ : ＜モジュール＞ | ＜ライブラリファイル＞[(＜ライブラリモジュール＞
                                  [,＜ライブラリモジュール＞]...)[,...]]
```

［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から除外するオブジェクトファイルを指定します。
  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】【V2.08.00 以降】」を参照してください。
- 【V3.00.00 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またはライブラリファイルを指定します。ライブラリファイルを指定した場合、ライブラリ内モジュール名を指定できます。
- 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無い場合はエラーとなります。

［例］

- a.obj, b.obj 及び c.lib 内の d モジュール内の関数を関数リストから除外します。

```
-cfi_ignore_module=a.obj,b.obj -cfi_ignore_module=c.lib(d)
```

-create_unfilled_area【V2.03.00 以降】

＜最適化リンケージエディタ（rlink）・オプション / 出力オプション＞

［指定形式］

```
-create_unfilled_area
```

［詳細説明］

- S レコード・ファイル (～ .mot) またはヘキサ・ファイル (～ .hex) を出力する際に、アセンブリ言語において .OFFSET 疑似命令を記述した箇所に作られる空き領域に対し、データを出力しないようにします。
- 本オプションの機能を利用するには、rlink で S レコード・ファイルおよびヘキサ・ファイルを作成するときだけではなく、ccrx コマンドまたは asrx コマンドでオブジェクト・ファイル (～ .obj) を作成するときにも指定する必要があります。

リストオプション

＜最適化リンケージエディタ（rlink）・オプション / リストオプション＞
リストオプションには、次のものがあります。

- -list
- -show

-list

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

［指定形式］

```
-list [= <ファイル名>]
```

［詳細説明］

- リストファイル出力およびリストファイル名を指定します。
- リストファイル名を指定しない場合には、出力（または先頭出力）ファイルと同じファイル名で、拡張子が form=library または extract 指定時「lbp」、それ以外のとき「map」のリストファイルが作成されます。
- -map オプションと同時指定した場合、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でも、リンケージ・リスト・ファイル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。【V2.06 以降】

-show

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

［指定形式］

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

［詳細説明］

- リストの出力内容を指定します。
- サブオプションの一覧を表 2.17 に示します。
- 各リストの具体例については「リンケージリスト」、「ライブラリリスト」を参照してください。

表 2.17 **show** オプションのサブオプション一覧

No	出力形式	サブオプション名	意味
1	form=library	symbol	モジュール内シンボル名一覧を出力します。
		reference	指定できません。
		section	モジュール内セクション一覧を出力します。
		xreference	指定できません。
		total_size	指定できません。
		vector	指定できません。
		relocation_attribute	指定できません。
		cfi	指定できません。
		all	モジュール内シンボル名、セクション一覧を出力します。

No	出力形式	サブオプション名	意味
2	form=library 以外	symbol	シンボルアドレス、サイズ、種別、最適化内容を出力します。
		reference	シンボルの参照回数を出力します。（form=relocate の時は指定できません）
		section	指定できません。
		xreference	クロスリファレンス情報を出力します。
		total_size	ROM 配置対象、RAM 配置対象ごとに、セクションの合計サイズを表示します。
		vector	ベクタ情報を出力します。（form=relocatable のときは指定できません）
		struct	構造体、共用体メンバ情報を出力します。（-form=rel/obj のときは指定できません）
		relocation_attribute	form=abs を指定し、かつ strip を指定していないとき、再配置属性を出力します。 form=hex/bin/stype を指定し、かつ入力ファイルが absolute/hex/stype 形式でないとき、再配置属性を出力します。 これら以外の場合は指定できません。
		cfi	関数リストを出力します。 オプション cfi 未指定時は指定できません（エラーになります）。 form=abs かつオプション strip 未指定時は指定できます。 form=hex/bin/stype かつ入力ファイルが absolute/hex/stype の場合は指定できます。 それ以外の場合は指定できません。
		all	show=symbol,xreferrence,total_size 指定時と同内容を出力します。（form=rel） show=symbol,reference,xreference,total_size,struct 指定時と同内容を出力します。（form=abs） show=symbol,reference,xreference,total_size,struct 指定時と同内容を出力します。（form=hex/stype/bin） form=obj のときは指定できません。

［備考］

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

		Symbol	Reference	Section	Xreference	Vector	Total_size	relocation_attribute	cfi
form=abs	show のみ	有効	有効	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	無効	無効
form=lib	show のみ	有効	無効	有効	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
form=rel	show のみ	有効	無効	無効	無効	無効	無効	無効	無効
	show=all	有効	無効	無効	有効[注]	無効	有効	無効	無効
form=obj	show のみ	有効	有効	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

		Symbol	Reference	Section	Xreference	Vector	Total_size	relocation_attribute	cfi
form=hex/bin/sty	show のみ	有効	有効	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]	有効[注]	無効	無効

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

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

最適化オプション

＜最適化リンケージエディタ（rlink）・オプション / 最適化オプション＞
最適化オプションには、次のものがあります。

- -optimize
- -nooptimize
- -samesize
- -symbol_forbid
- -samecode_forbid
- -section_forbid
- -absolute_forbid
- -ALLOW_OPTIMIZE_ENTRY_BLOCK【V3.06.00 以降】

-optimize

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

［指定形式］

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

- 省略時解釈
  optimize です。

［詳細説明］

- コンパイル、アセンブル時に goptimize オプションを指定したファイルに対して最適化を行います。
- サブオプションがない場合は、すべての最適化を実行します。
  -optimize=symbol_delete,same_code,short_format,branch と同じ意味を持ちます。
- -optimize=speed は、オブジェクトスピード低下を招く可能性のある最適化以外を実行します。
  -optimize=symbol_delete,short_format,branch と同じ意味を持ちます。
- -optimize=safe は、変数や関数の属性によって制限される可能性のある最適化以外を実行します。
  -optimize=short_format,branch と同じ意味を持ちます。
- その他のサブオプションが意味する最適化の内容は、次の表の通りです。

表 2.18 **optimize** オプションのサブオプション一覧

サブオプション	意 味	最適化対象プログラム[注 1]	
		RXC	RXA
symbol_delete	1 度も参照のない変数 / 関数を削除します。必ずコンパイル 時に #pragma entry を指定するか、rlink で entry オプションを指定してください。	○	×
same_code	複数の同一命令列をサブルーチン化します。	○	×
short_format	ディスプレースメント / イミディエートのコードサイズを短縮可能な場合、コードサイズがより小さくなる命令に置き換えます。	○	○
branch	プログラムの配置情報に基づいて、分岐命令サイズを最適化します。symbol_delete 以外の他の最適化項目を実行すると、指定の有無に関わらず必ず実行します。	○	○

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

［備考］

- form=｛object|relocate|library｝または strip 指定時、本オプションは無効です。
- #pragma entry または entry オプションで実行開始アドレスを指定していない場合、optimize=symbol_delete は無効になります。

-nooptimize

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

［指定形式］

```
-nooptimize
```

- 省略時解釈
  optimize です。

［詳細説明］

- リンク時最適化を行いません。

-samesize

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

［指定形式］

```
-samesize = <サイズ>
```

- 省略時解釈
  samesize=1E です。

［詳細説明］

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

［備考］

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

-symbol_forbid

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

［指定形式］

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

［詳細説明］

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

［備考］

- 最適化を使用しないリンク処理では、本オプションは無効です。

-samecode_forbid

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

［指定形式］

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

［詳細説明］

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

［備考］

- 最適化を使用しないリンク処理では、本オプションは無効です。

-section_forbid

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

［指定形式］

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

［詳細説明］

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

［備考］

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

-absolute_forbid

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

［指定形式］

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

［詳細説明］

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

［備考］

- 最適化を使用しないリンク処理では、本オプションは無効です。

-ALLOW_OPTIMIZE_ENTRY_BLOCK【V3.06.00 以降】

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

［指定形式］

```
-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。

［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。

セクションオプション

＜最適化リンケージエディタ（rlink）・オプション / セクションオプション＞
セクションオプションには、次のものがあります。

- -start
- -fsymbol
- -aligned_section

-start

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

［指定形式］

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

- 省略時解釈
  セクションを 0 番地から割り付けます。

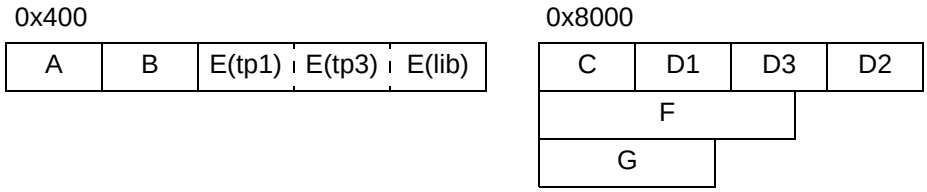
［詳細説明］

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

［例］

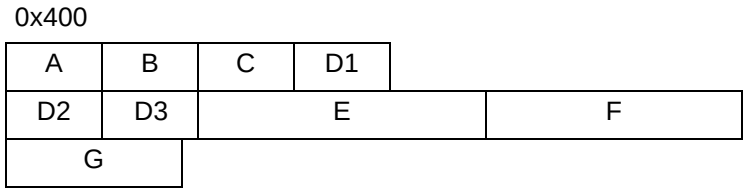
下記順番でオブジェクトを入力する場合のセクション配置を例に示します。
（括弧内は各オブジェクトが持つセクション）
tp1.obj(A,D1,E) -> tp2.obj(B,D3,F) -> tp3.obj(C,D2,E,G) -> lib.lib(E)

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



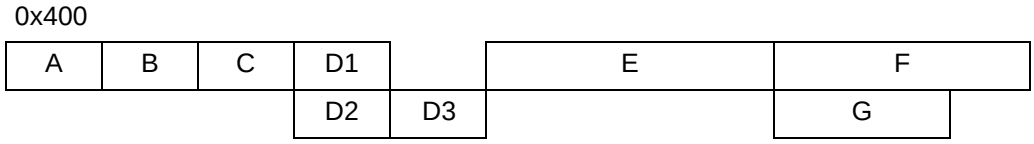
- ":" で区切った C,F,G セクションは、同一アドレスに割り付きます。
- ワイルドカードで記述したセクション（ここでは D で始まる名前のセクション）は、
  入力した順番で割り付きます。
- 同名セクション内（ここでは E セクション）は、入力したオブジェクトから順番に割り付きます。
- ライブラリ入力による同名セクション（ここでは E セクション）は、入力オブジェクトの次に割り付きます。

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



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

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



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

［備考］

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

-fsymbol

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

［指定形式］

```
-fsymbol = < セクション名 >[,…]
```

［詳細説明］

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

［例］

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

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

- [test.fsy の出力例]

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

［備考］

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

-aligned_section

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

［指定形式］

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

［詳細説明］

- 指定セクションのアライメント数を 16 byte に変更します。

［備考］

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

ベリファイオプション

＜最適化リンケージエディタ（rlink）・オプション / ベリファイオプション＞
ベリファイオプションには、次のものがあります。

- -cpu
- -contiguous_section

-cpu

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

［指定形式］

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

［詳細説明］

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

［例］

サブオプション stride を指定しない場合

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

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

サブオプション stride を指定した場合

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

- D1,D2 が ROM 属性の領域に（セクションを分割して / 分割しないで）収まるとき、正常終了します。セクションを分割しても収まらないときリンクエラーになります。
- セクション割り付けが可能なアドレス範囲を 16 進数で指定してください。ROM/RAM の属性は、モジュール間最適化で使用します。
- メモリ種別 “FIX” には、アドレス固定の領域（I/O エリア等）を指定します。
- メモリ種別 “FIX” と、それ以外のメモリ種別のアドレス範囲が重複した場合は、メモリ種別 “FIX” を有効とします。
- サブオプション stride は、メモリ種別が、ROM または RAM で、アドレス範囲にセクションが収まらなかった場合に、セクションを分割して同じメモリ種別の領域に割り付けます。
- サブオプション stride で、セクションを分割する単位は、モジュール単位になります。

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

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

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

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

［備考］

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

-contiguous_section

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

［指定形式］

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

［詳細説明］

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

［例］

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

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

［備考］

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

その他オプション

＜最適化リンケージエディタ（rlink）・オプション / その他オプション＞
その他オプションには、次のものがあります。

- -s9
- -stack
- -compress
- -nocompress
- -memory
- -rename
- -lib_rename 【V3.01.00 以降】
- -delete
- -replace
- -extract
- -strip
- -change_message
- -hide
- -total_size
- -verbose 【V3.03.00 以降】

-s9

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

［指定形式］

```
-s9
```

［詳細説明］

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

［備考］

- form=stype 指定がないとき、本オプションは無効です。

-stack

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

［指定形式］

```
-stack
```

［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 になります。

［備考］

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

-compress

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

［指定形式］

```
-compress
```

- 省略時解釈
  デバッグ情報を圧縮しません。

［詳細説明］

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

［備考］

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

-nocompress

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

［指定形式］

```
-nocompress
```

- 省略時解釈
  デバッグ情報を圧縮しません。

［詳細説明］

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

-memory

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

［指定形式］

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

- 省略時解釈
  memory=high です。

［詳細説明］

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

［備考］

- 下記オプションを指定した場合、-memory=low オプション指定は無効となります。
  - form=absolute,hexadecimal,stype,binary 指定時
    compress,delete,rename,map,stack,cpu=stride
    list と show[={reference | xreference}] を同時指定
  - form=library 指定時
    delete,rename,extract,hide,replace,allow_duplicate_module_name
  - form=object,relocate 指定時
    extract
  - optimize を指定時
- また、入力ファイルや出力ファイル形式によっても無効となる組み合わせがあります。

-rename

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

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-lib_rename 【V3.01.00 以降】

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

［指定形式］

```
-lib_rename = < 名前 1>=< 名前 2>[,…]
-lib_rename = < ファイル >(< 名前 1>=< 名前 2>[,…])
-lib_rename = "< ファイル >|< モジュール >[|< モジュール >…](< 名前 1>=< 名前 2>[,…])"
```

［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。
- < 名前 1> には変更対象のシンボル名、またはセクション名、< 名前 2> 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、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 < 名前 >、< ファイル >、< モジュール > がみつからない場合。
  - パラメータを省略した場合。

［例］

- b.lib 及び c.lib にある _sym1 を _data に変更します。

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

- b.lib にある全てのモジュール内の _sym1 を _data に変更します。

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

- b.lib にあるモジュール m1 及び m2 内の _sym1 を _data に変更します。

```
> rlink a.obj -lib=b.lib,c.lib  -lib_rename="b.lib|m1|m2(_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。また、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 -merge_files と本オプションを組み合わせて使用した場合、動作は保証されません。

-delete

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

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-replace

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

［指定形式］

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

［詳細説明］

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

［例］

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

［備考］

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

-extract

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

［指定形式］

```
-extract = < モジュール名 >[,…]
```

［詳細説明］

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

［例］

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

［備考］

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

-strip

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

［指定形式］

```
-strip
```

［詳細説明］

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

［例］

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

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

［備考］

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

-change_message

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

［指定形式］

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

［詳細説明］

- インフォメーション、ウォーニング、エラーレベルのメッセージレベルを変更します。
- メッセージ出力時の実行継続 / 中断を変更できます。
- エラー番号を指定すると、指定した番号のメッセージの種別を変更します。
- また、ハイフン（-）を使用して、メッセージ番号の範囲を指定することもできます。
- エラー番号には、コンポーネント番号（05）、発生フェーズ（6）に続けて出力される 4 桁の数値（E0562310 の場合は 2310）を指定します。
- メッセージ番号の指定を省略した場合は、すべてのメッセージの種別を指定したものに変更します。

［例］

```
change_message=warning=2310
```

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

```
change_message=error
```

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

-hide

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

［指定形式］

```
-hide
```

［詳細説明］

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

［例］

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

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

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

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

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

［備考］

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

-total_size

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

［指定形式］

```
-total_size
```

［詳細説明］

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

［備考］

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

-verbose 【V3.03.00 以降】

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

［指定形式］

```
-verbose=<sub>[, ...]
sub : crc
```

［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。

［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
```

サブコマンドファイルオプション

＜最適化リンケージエディタ（rlink）・オプション / サブコマンドファイルオプション＞
サブコマンドファイルオプションには、次のものがあります。

- -subcommand

-subcommand

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

［指定形式］

```
-subcommand = < ファイル名 >
```

［詳細説明］

- オプションをサブコマンドファイルで指定します。
- サブコマンドファイルの書式は以下の通りです。
- < オプション > {= | △ } ［< サブオプション > ［,…］][△ &] [;< コメント >]
- オプションとサブオプションの区切りは、= の代わりに空白も指定できます。
- input オプションの場合は、サブオプション区切りに空白を指定できます。
- サブコマンドファイル内では 1 オプション / 行で指定します。
- サブオプションを 1 行に記述できない場合は、& を用いて継続指定できます。

［例］

- コマンドライン指定

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

- サブコマンド指定

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

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

残りのオプション

＜最適化リンケージエディタ（rlink）・オプション / 残りのオプション＞
残りのオプションには、次のものがあります。

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

-logo

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

［指定形式］

```
-logo
```

- 省略時解釈
  コピーライト表示を出力します。

［詳細説明］

- コピーライト表示を出力します。

-nologo

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

［指定形式］

```
-nologo
```

- 省略時解釈
  コピーライト表示を出力します。

［詳細説明］

- コピーライト表示出力を抑止します。

-end

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

［指定形式］

```
-end
```

［詳細説明］

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

［例］

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

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

-exit

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

［指定形式］

```
-exit
```

［詳細説明］

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

［例］

- コマンドライン指定

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

- test.sub

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

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

2.5.4 ライブラリジェネレータ・オプション

ライブラリ生成フェーズのオプションの分類と説明を以下に示します。

分類 .	オプション	説明
ライブラリオプション	-head	構築対象のライブラリを指定
	-output	出力ライブラリファイル名を指定
	-nofloat	簡易入出力関数の生成
	-reent【V2.03.00 以降】	リエントラントライブラリの生成
	-lang	使用可能な C 言語標準ライブラリ関数構成の選択
	-simple_stdio	機能縮小版入出力関数の生成
	-secure_malloc【Professional 版のみ】【V2.05.00 以降】	セキュリティ機能を追加した calloc、free、malloc、realloc を生成
	-logo -nologo	コピーライトを出力 コピーライトの出力を抑止

また、ライブラリジェネレータにはコンパイル・フェーズのオプションも指定することができます。指定したオプションは、ライブラリジェネレータがライブラリをコンパイルする際のコンパイルオプションとして利用されます。ただし、指定しても無視されるものや、特定の指定値に固定されるものがあります。ライブラリジェネレータに指定したコンパイルオプションがどのように解釈されるかについては表 2.19 を参照してください 。

ライブラリオプション

＜ライブラリジェネレータ・オプション / ライブラリオプション＞
ライブラリオプションには、次のものがあります。

- -head
- -output
- -nofloat
- -reent【V2.03.00 以降】
- -lang
- -simple_stdio
- -secure_malloc 【Professional 版のみ】【V2.05.00 以降】
- -logo
- -nologo

-head

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

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

- 省略時解釈
  head=all です。

［詳細説明］

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

-output

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-output=< ファイル名 >
```

- 省略時解釈
  output=stdlib.lib です。

［詳細説明］

- 出力ファイル名を指定します。

-nofloat

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-nofloat
```

［詳細説明］

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

［備考］

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

-reent【V2.03.00 以降】

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-reent
```

［詳細説明］

- リエントラントライブラリを生成します。ただし、EC++ ライブラリ関数、および rand、srand 関数はリエントラントにはなりません。

［備考］

- リエントラントライブラリをリンクする場合は、プログラム内で標準インクルードファイルをインクルードする前に _REENTRANT というマクロ名を #define で定義するか、コンパイル時に define オプションで _REENTRANT を定義してください。

-lang

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

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

- 省略時解釈
  lang=c です。

［詳細説明］

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

［備考］

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

-simple_stdio

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-simple_stdio
```

［詳細説明］

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

［備考］

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

-secure_malloc 【Professional 版のみ】 【V2.05.00 以降】

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-secure_malloc
```

［詳細説明］

記憶域に対する不正な操作を検出するためのセキュリティ機能を追加した calloc、free、malloc、realloc 関数を生成します。
次の操作を行なった場合に __heap_chk_fail 関数を呼び出します。

- calloc、malloc、realloc で確保し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 で確保した領域の外側（前後それぞれ 4 バイトまで）に値を書き込んだ後 free、realloc に渡す。

C++ プログラムの new、delete 演算子にも同じ機能が追加されます。
__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。

- 返却値の型は void 型のみであり、仮引数を持たない関数です。
- C++ プログラム内で __heap_chk_fail 関数を定義する場合は「extern "C"」を付加してください。
- __heap_chk_fail 関数内で再帰的に記憶域の破壊を検出しないように注意してください。
- __heap_chk_fail 関数を定義する場合は、static を指定しないでください。

［記述例］

```
#include <stdlib.h>

void sub(int *ip) {
    ...
    free(ip);
}

int func(void) {
    int *ip;
    if ((ip = malloc(40 * sizeof(int))) == NULL)
        if ((ip = malloc(10 * sizeof(int))) == NULL) return(1);
        else sub(ip); /* 1 回目の free */
    else
        ...
    free(ip); /* 2 回目の free */
    return(0);
}

#ifdef __cplusplus
extern "C" {
#endif
void __heap_chk_fail(void) {
    /* ヒープ領域破壊を検出したときの処理 */
}
#ifdef __cplusplus
}
#endif
```

［備考］

セキュリティ機能を追加した 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 4 バイト余分に領域を確保します。このため通常より多くヒープ領域を消費します。これは C++ プログラムの new 演算子も同じです。

-logo

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-logo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

- コピーライト表示が出力されます。

-nologo

＜ライブラリジェネレータ・オプション / ライブラリオプション＞

［指定形式］

```
-nologo
```

- 省略時解釈
  コピーライト表示が出力されます。

［詳細説明］

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

無効となるコンパイラオプション

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

表 2.19 無効オプション一覧

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
1	include	常に無効	なし
2	define	常に無効	なし
3	undefined	常に無効	なし
4	message nomessage	常に無効	nomessage
5	change_message	常に無効	なし
6	file_inline_path	常に無効	なし
7	comment	常に無効	なし
8	check	常に無効	なし
9	output	常に無効	output=obj
10	noline	常に無効	なし
11	debug nodebug	常に無効	nodebug
12	listfile nolistfile show	常に無効	nolistfile
13	file_inline	常に無効	なし
14	asmcmd	常に無効	なし
15	lnkcmd	常に無効	なし
16	asmopt	常に無効	なし
17	lnkopt	常に無効	なし
18	logo nologo	常に無効	nologo
19	euc sjis latin1 utf8	常に無効	なし
20	outcode	常に無効	なし
21	subcommand	常に無効	なし
22	alias	常に無効	alias=noansi
23	pic pid	lang=cpp または C++ ソースコンパイル時[注 1]	なし
24	ip_optimize	常に無効	なし

No.	無効とされるオプション	無効になる条件	無効になった場合にライブラリ構築時に選択されるオプション
25	merge_files	常に無効	なし
26	whole_program	常に無効	なし
27	big5 gb2312	常に無効[注 2]	なし
28	map 【V3.02.00 以降】	常に無効[注 3]	smap
29	control_flow_integrity	常に無効	なし
30	create_unfilled_area 【V3.00.00 以降】	常に無効	なし
31	stack_protector 【V3.00.00 以降】	常に無効	なし
32	stack_protector_all 【V3.00.00 以降】	常に無効	なし
33	misra2004	常に無効	なし
34	misra2012 【V3.00.00 以降】	常に無効	なし
35	misra_intermodule 【V3.01.00 以降】	常に無効	なし
36	tfu 【V3.01.00 以降】	常に無効	なし
37	truncated_address_initializer 【V3.01.00 以降】	常に無効	なし
38	g_line 【V3.02.00 以降】	常に無効	なし
39	tfu_version 【V3.05.00 以降】	常に無効	なし
40	nosave_tfu 【V3.05.00 以降】	常に無効	なし

注 1. 警告 W0511171 が表示されます。

注 2. エラー F0593305 になります (ライブラリ生成できません)。

注 3. < ファイル名 > の指定があっても無視します。< ファイル名 > が存在しなくてもエラーを出力しません。

3. 出力ファイル

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

3.1 アセンブル・リスト・ファイル

アセンブラが出力するアセンブル・リスト・ファイルの内容と形式について説明します。
ソースリストファイルには、コンパイル結果およびアセンブル結果の情報を表示します。
ソースリストの構成と内容を示します。

表 3.1 ソースリストの構成と内容

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

注 -listfile オプションを指定した場合に有効です。

3.1.1 ソース情報

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

3.1.2 オブジェクト情報

オブジェクト情報の出力例を示します。

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

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

                                 ;*** CPU TYPE ***

                                 ;-ISA=RXV1

                                 ;*** COMMAND PARAMETER ***

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

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

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

```
0000001D 3F6702                 RTSD #08H, R6-R7
00000020                        _func03:
                                .STACK_func03=4
                                ;       15
                                ;       16 int func03(int p)
                                ;       17 {
                                ;       18     return p+1;
00000020 6211                   ADD #01H, R1
                                ;       19 }
00000022 02                     RTS
                                .SECTIOND,ROMDATA,ALIGN=4
00000000                        _y:
00000000 01000000               .lword00000001H
                                .END
```

項番	説明
(1)	ロケーション情報（LOC.） アセンブル時に決定できる範囲のオブジェクトコードのロケーションアドレスを出力します
(2)	オブジェクトコード情報（OBJ.） ニーモニックに対応するオブジェクトコードを出力します
(3)	行情報（0XMDA） アセンブラがソースを処理した結果の情報を出力します。 各記号の意味を下記に示します。

0	X	M	S	D	内　容
0-30					インクルードファイルのネストレベルを示します。
	X				-show=conditionals 指定時、条件アセンブルで条件が偽となった行を示します。
		M			-show=expansions 指定時、マクロ命令の展開行であることを示します。
		D			-show=definitions 指定時、マクロ命令の定義行であることを示します。
			S		分岐距離指定子 S を指定したことを示します。
			B		分岐距離指定子 B を指定したことを示します。
			W		分岐距離指定子 W を指定したことを示します。
			A		分岐距離指定子 A を指定したことを示します。
				*	条件分岐命令に対して代替命令を選択したことを示します。

(4)	ソース情報（SOURCE STATEMENT） アセンブリソースファイルの内容を表示します
(5)	C/C++ ソース行番号（LineNo.）
(6)	C/C++ ソース（C-SOURCE STATEMENT） -show=source オプションを指定した場合、C/C++ ソースを出力します

3.1.3 統計情報

統計情報の出力例を示します。

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

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

3.1.4 コンパイラのコマンド指定情報

コンパイラを起動したときのコマンドで指定されたファイル名とオプションを表示します。
コンパイラのコマンド指定情報は、リストファイルの先頭に出力されます。
コマンド指定情報の出力例を示します。

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

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

3.1.5 アセンブラのコマンド指定情報

アセンブラを起動したときのコマンドで指定されたファイル名とオプションを表示します。
アセンブラのコマンド指定情報は、リストファイルの最後に出力されます。
コマンド指定情報の出力例を示します。

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

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

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 リンケージリストの構成

リンケージリストの構成と内容を表 3.2 に示します。

表 3.2 リンケージリストの構成と内容

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

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

3.2.2 オプション情報

コマンドライン、サブコマンドファイルで指定したオプション列を出力します。
オプション情報の出力例を示します（rlink -subcommand=test.sub -list -show 指定時）。

```
(test.sub の内容)
INPUT test.obj
```

```
*** Options ***
-sub=test.sub      (1)
INPUT test.obj     (2)
-list              (1)
-show              (1)
```

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

3.2.3 エラー情報

エラーメッセージを出力します。エラー情報の出力例を示します。

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

項番	説明
(1)	エラーメッセージを出力します

3.2.4 リンケージマップ情報

各セクションの先頭／最終アドレス、サイズ、種別をアドレス順に出力します。
リンケージマップ情報の出力例を示します。

```
*** Mapping List ***
(1)                              (2)       (3)       (4)    (5)    (6)
SECTION                          START     END       SIZE   ALIGN  ATTRIBUTE
P
                                 00001000  00001000     1   1      CODE
C
                                 00001004  00001007     4   4      ROMDATA
D_2
                                 00001008  000014dd    4d6  2      ROMDATA
B_2
                                 000014de  000050b3   3bd6  2      DATA
```

項番	説明
(1)	セクション名を表示します
(2)	先頭アドレスを表示します **OVER** と表示された場合は、アドレスが 32bit で表現できる範囲を超えたことを示しています
(3)	最終アドレスを表示します **OVER** と表示された場合は、アドレスが 32bit で表現できる範囲を超えたことを示しています
(4)	セクションサイズを表示します
(5)	セクションのアライメント数を表示します
(6)	再配置属性を表示します

注　リンク後にアドレスが確定したコードセクションのアライメント数は、big-endian を選択した場合、コンパイルおよびアセンブル時のアライメント数に関わらず 4 の倍数で表示されます。

3.2.5 シンボル情報

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

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

項番	説明
(1)	セクション名を表示します
(2)	ファイル名を表示します
(3)	(2) のファイルに含まれる該当セクションの先頭アドレスを表示します
(4)	(2) のファイルに含まれる該当セクションの最終アドレスを表示します
(5)	(2) のファイルに含まれる該当セクションのセクションサイズを表示します
(6)	シンボル名を表示します ただし、__ を含むシンボル名は名前が変わる場合があります
(7)	シンボルアドレスを表示します **OVER** と表示された場合は、アドレスが 32bit で表現できる範囲を超えたことを示しています
(8)	シンボルサイズを表示します
(9)	シンボル種別を次のように表示します データ種別 func: 関数名 data: 変数名 entry: エントリ関数名 none: 未設定（ラベル、アセンブラシンボル） 宣言種別 g: 外部定義 l: 内部定義
(10)	シンボル参照回数を表示します -show=reference を指定した場合のみ表示します 参照回数を表示しないときは、* を表示します
(11)	最適化有無を次のように表示します ch: 最適化によって変更されたシンボル cr: 最適化によって生成されたシンボル mv: 最適化によって移動されたシンボル

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

```
*** Symbol List ***

SECTION=
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=tp.obj
                                   00000000  0000000b         c
  _st
                                   00000000         c   data ,g         0
    struct  {
                                                    c
      _st.mem1
                                   00000000         1   char
      _st.mem2
                                   00000004         4   int
      _st.mem3
                                   00000008         2   short
    }
```

項番	説明
(1)	構造体の場合は struct、共用体の場合は union
(2)	構造体または共用体全体のサイズ
(3)	メンバ名をシンボル名に “.”（ドット）で連結して表示
(4)	メンバアドレスを表示
(5)	メンバのサイズを表示
(6)	メンバの型を表示

3.2.6 シンボル削除最適化情報

シンボル削除最適化（-optimize=symbol_delete）によって削除されたシンボルのサイズ、種別を出力します。
シンボル削除最適化情報の出力例を示します。

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

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

3.2.7 クロスリファレンス情報

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

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

項番	説明
(1)	Unit 番号。オブジェクト単位の識別番号

項番	説明
(2)	オブジェクト名。 リンク時の入力指定順になる
(3)	シンボル名。セクションごとに配置アドレスの昇順に出力される
(4)	シンボルの配置アドレス。 -form=relocate 指定時は、セクション先頭からの相対値となる **OVER** と表示された場合は、アドレスが 32bit で表現できる範囲を超えたことを示す
(5)	参照している外部シンボルのアドレスを表す。 出力形式は以下のようになる。 <Unit 番号 >(< アドレス or セクション内オフセット >:< セクション名 >)

3.2.8 合計セクションサイズ

ROM セクション、RAM セクション、およびプログラムセクションの合計サイズを出力します。
合計の出力例を示します。

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

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

3.2.9 ベクタ情報

-show=vector を指定した場合、可変ベクタテーブルの内容を表示します。
合計の出力例を示します。

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

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

3.2.10 CRC 情報

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

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

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

3.2.11 CFI 情報

-show=cfi を指定した場合、不正な間接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 Function 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。

3.3 ライブラリ・リスト

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

3.3.1 ライブラリリストの構成

ライブラリリストの構成と内容を示します。

表 3.3 ライブラリリストの構成と内容

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

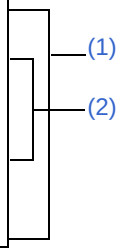
注 すべてのオプションは、-list オプションを指定した場合に有効です。

3.3.2 オプション情報

コマンドライン、サブコマンドファイルで指定したオプション列を出力します。
オプション情報の出力例を示します（rlink -subcommand=test.sub -list -show を指定した場合）。

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

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



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

3.3.3 エラー情報

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

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

項番	説明
(1)	ウォーニングメッセージを出力します。

3.3.4 ライブラリ情報

ライブラリの種別を出力します。ライブラリ情報の出力例を示します。

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

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

3.3.5 ライブラリ内モジュール、セクション、シンボル情報

ライブラリ内のモジュール一覧を出力します。
-show=symbol を指定した場合はモジュール内シンボル名一覧を、-show=section を指定した場合はモジュール内セクション名、シンボル名一覧を出力します。
ライブラリ内モジュール、セクション、シンボル情報の出力例を示します。

```
***  List ***
(1)             (2)
MODULE          LAST UPDATE
  (3)
  SECTION
    (4)
    SYMBOL

adhry
                29-Feb-2000 12:34:56
  P
    _main
    _Proc0
    _Proc1
  C
  D
    _Version
  B
    _IntGlob
    _CharGlob
```

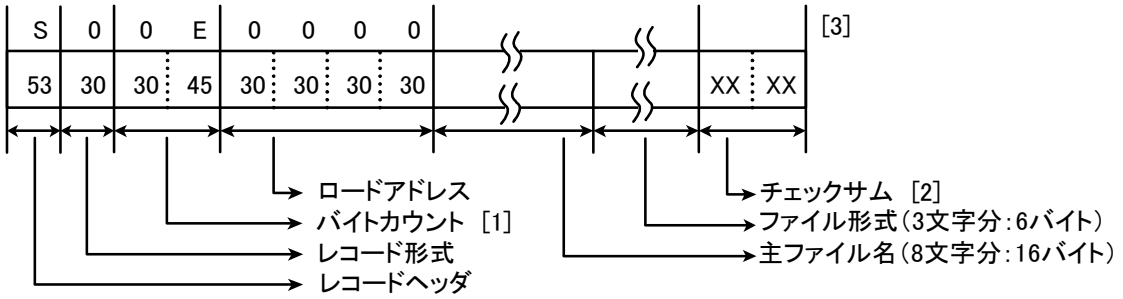
項番	説明
(1)	モジュール名を表示します。
(2)	モジュールを登録した日付を表示します。 モジュールが更新された場合は、最新の更新日付を表示します。
(3)	モジュール内セクション名を表示します。
(4)	セクション内をシンボル表示します。

3.4 モトローラ S 形式、インテル HEX 形式ファイル

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

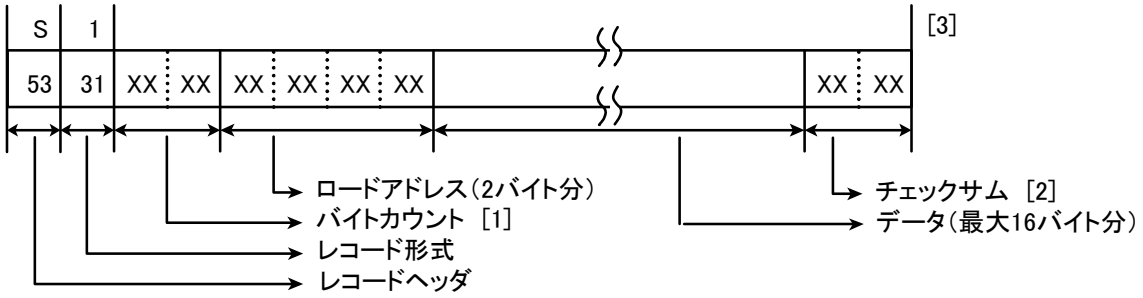
3.4.1 モトローラ S 形式ファイル

(a) ヘッダレコード(S0レコード)

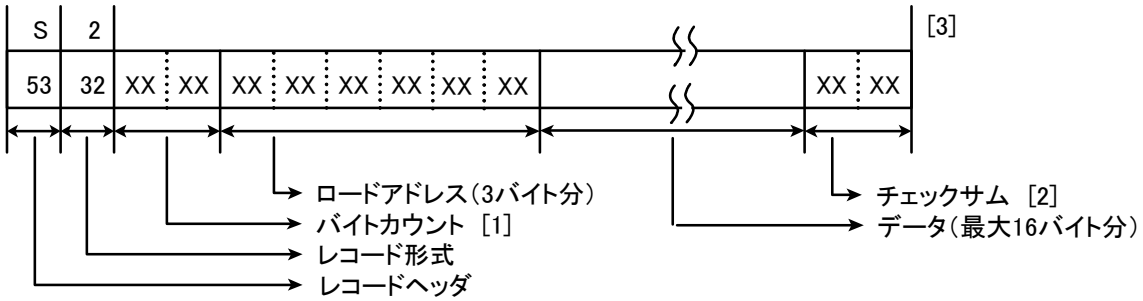


(b) データレコード(S1, S2, S3レコード)

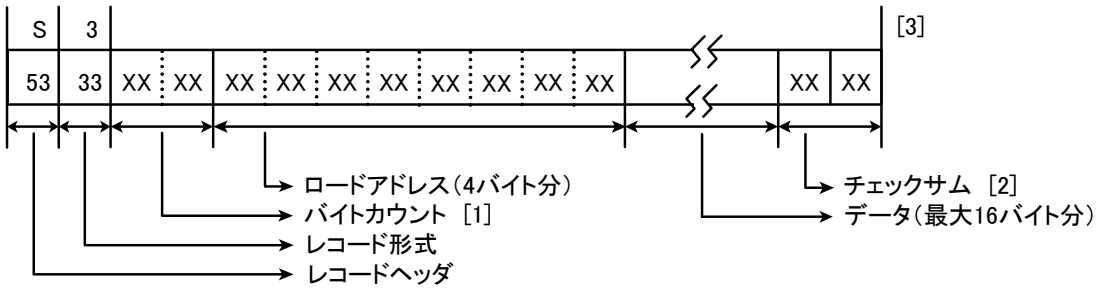
(i) ロードアドレスが0～FFFFの場合



(ii) ロードアドレスが10000～FFFFFFの場合

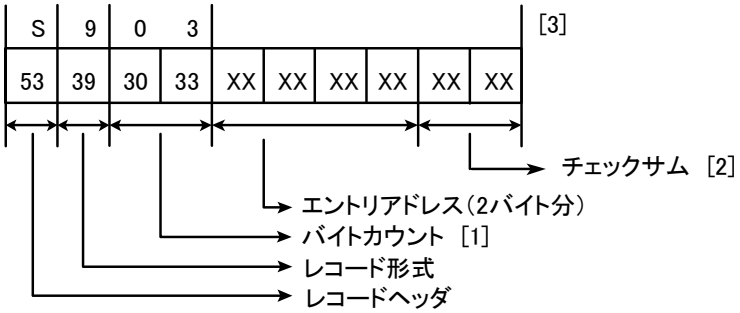


(iii) ロードアドレスが1000000～FFFFFFFFの場合

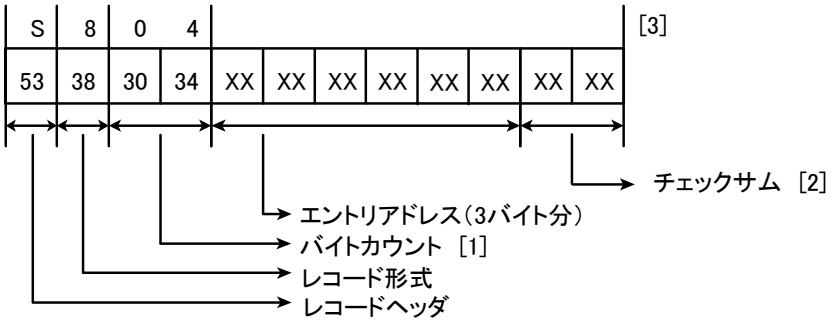


(c) エンドレコード(S9, S8, S7レコード)

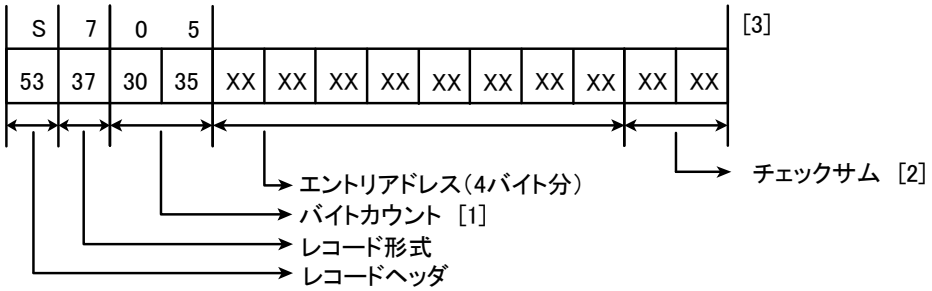
(i) エントリアドレスが0～FFFFの場合



(ii) エントリアドレスが10000～FFFFFFの場合



(iii) エントリアドレスが1000000～FFFFFFFFの場合



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

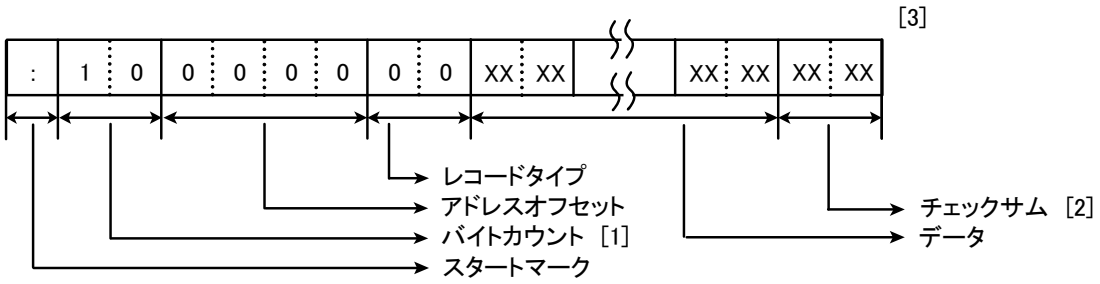
3.4.2 インテル HEX 形式ファイル

各データレコードの実行アドレスは以下のように求めます。

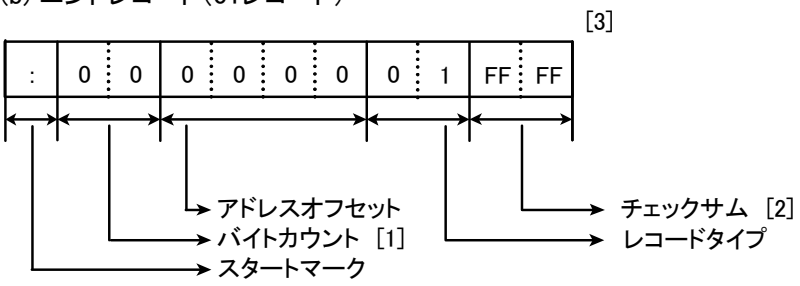
(1) セグメントアドレスの場合
(セグメントベースアドレス << 4) + (データレコードのアドレスオフセット)

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

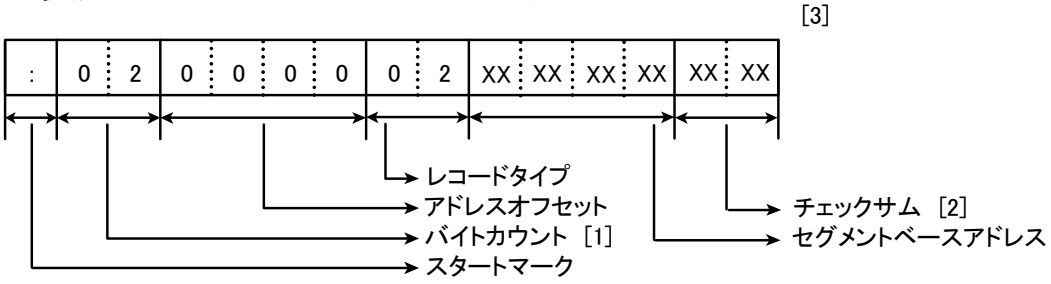
(a) データレコード(00レコード)



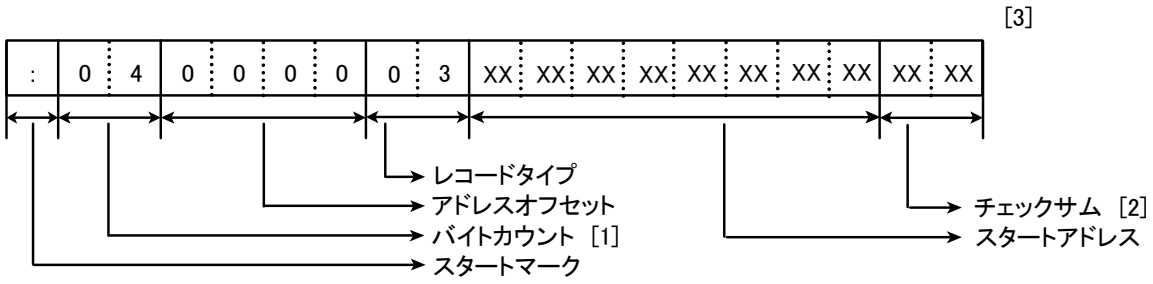
(b) エンドレコード(01レコード)



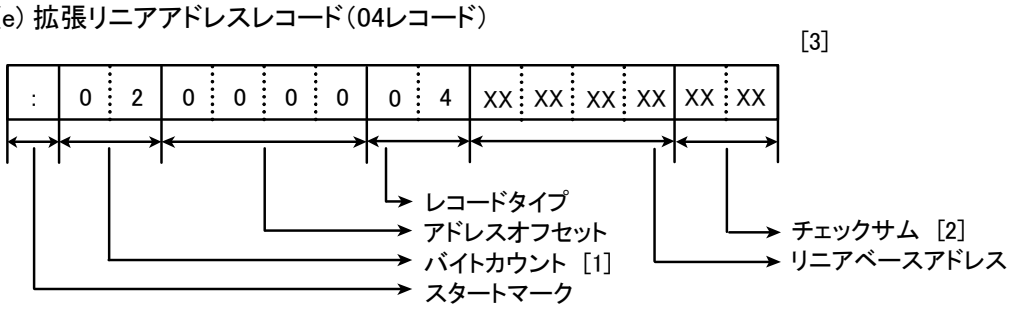
(c) 拡張セグメントアドレスレコード(02レコード)



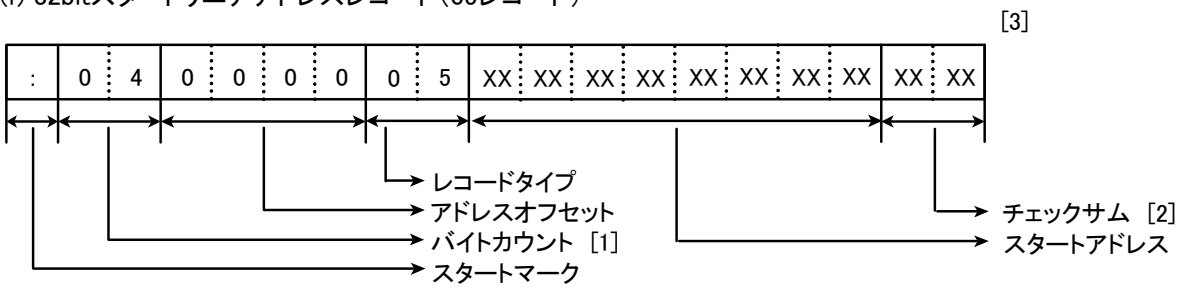
(d) スタートアドレスレコード(03レコード)



(e) 拡張リニアアドレスレコード(04レコード)



(f) 32bitスタートリニアアドレスレコード(05レコード)



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

4. コンパイラ言語仕様

4.1 基本言語仕様

CC-RX は、ANSI 規格で規定された言語仕様をサポートしていますが、その中には処理系定義として規定されている項目があります。ここでは、RX マイクロコントローラの処理系に依存した項目の言語仕様について説明します。
なお、CC-RX で独自に追加されている拡張言語仕様については、「4.2 拡張言語仕様」を参照してください。

4.1.1 未規定の動作

この項では、ANSI 規格における未規定の動作項目について説明します。

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

(2) 文字表示の意味ー後退（**\b**）、水平タブ（**\t**）、垂直タブ（**\v**）
表示装置設計依存となります。

(3) 型ー浮動小数点
IEEE754 注準拠です。

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

(4) 式ー評価順序
基本的には式は前方より評価します。ただし、最適化を行った場合は未規定とします。オプションなどにより、順序が変更になることがあるので、副作用のある式の記述は行わないでください。

(5) 関数呼び出しー引数の評価順序
基本的には第一引数（先頭の引数）より順に評価します。ただし、最適化を行った場合は未規定とします。オプションなどにより、順序が変更になることがあるので、副作用のある式の記述は行わないでください。

(6) 構造体指定子、および共用体指定子
ビットフィールドの型の整列境界をまたがないように調整します。オプションや #pragma 指令でパッキングを行った場合は、整列境界調整は行わず、ビットフィールドは詰めて配置されます。

(7) 関数定義ー 仮引数の記憶域
スタック、およびレジスタに割り付けます。詳細は、「9.1.3 引数の設定、参照に関する規則」を参照してください。

(8) # 演算子
前方より評価します。

4.1.2 未定義の動作

この項では、ANSI 規格における未定義の動作項目について説明します。

(1) 文字集合
ソースファイル中に文字集合に定められた文字以外がある場合、メッセージを出力します。

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

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

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

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

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

(7) 文字列リテラルー結合
単純文字列リテラルとワイド文字列リテラル字句が隣り合うとき、単純に文字列結合を行います。

(8) 文字列リテラルー変更
文字列リテラルの変更はユーザ責任となります。RAM に配置した場合は変更されますが、ROM に配置された場合は変更されません。

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

(10) 浮動小数点型と汎整数型
浮動小数点型の値を汎整数型に型変換する場合、整数部の値が汎整数型で表現できなければ、汎整数型で表現できる値に切りつめを行います。

(11) 左辺値及び関数指示子
不完全型が左辺値となった場合は、メッセージを出力します。

(12) 関数呼び出しー引数の個数
実引数の数が少ない場合、仮引数は不定値となります。実引数が多い場合、余剰な実引数はないものとして関数が実行され、実引数は意味を持ちません。
関数呼び出しに先立ち、関数宣言がある場合は、メッセージを出力します。

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

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

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

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

(17) キャスト演算子ー関数ポインタのキャスト
型変換されたポインタが元の型以外の関数を呼び出すために使われた場合、関数を呼び出すことは可能です。引数、戻り値が不適合な場合は、不正となります。

(18) キャスト演算子ー汎整数型のキャスト
ポインタを汎整数型にキャストした場合で、領域の大きさが不十分な場合は、キャストした型の領域の大きさに切り詰められます。

(19) 乗除演算子
コンパイル中に 0 による除算剰余算を検出した場合、メッセージを出力します。
実行時は、0 除算例外が発生します。エラー処理ルーチンを記述した場合は、それに従います。

(20) 加減演算子ー配列以外のポインタ
配列オブジェクトの要素を指すかのように動作するもの以外のポインタに対して、加算、または減算を行っている場合、あたかも配列の要素を指しているように振舞います。

(21) 加減演算子ー別な配列へのポインタ減算
同じ配列オブジェクトの中を指すかのように動作するもの以外の二つのポインタに対し、減算を行なっている場合、あたかも配列の要素を指しているように振舞います。

(22) ビット単位のシフト演算子
右オペランドの値が負であるか、または拡張した左オペランドのビット幅以上の場合、左オペランドのビット幅で右オペランドをマスクした値でシフトした値となります。

(23) 関係演算子ーポインタ
比較対象のポインタで指されているオブジェクトが同一の集成体オブジェクト、または共用体オブジェクトのメンバでない場合、あたかも同一の集成体オブジェクト、または共用体オブジェクトのメンバを指しているポインタ同士の関係演算であるかのように比較します。

(24) 単純代入
オブジェクトに格納されている値が、何らかの形でそのオブジェクトの記憶域に重なる他のオブジェクトを通してアクセスされる場合、重なりは完全に一致していなければなりません。さらに、二つのオブジェクトの型は、

適合する型の修飾版、または非修飾版でなければなりません。一致しない重なりの代入は、代入によって代入元の値が破壊されます。

(25) 構造体指定子及び共用体指定子
メンバ宣言並びが名前付のメンバを含まない場合、C 言語としてコンパイルすると、意味を持たない旨の警告メッセージを出力します。C++ 言語としてコンパイルすると、C 言語と同一メッセージでエラーとします。

(26) 型修飾子ー const
メンバ宣言並びが名前付のメンバを含まない場合、意味を持たない旨の警告メッセージを出力します。

(27) 型修飾子ー volatile
volatile 修飾型で定義されたオブジェクトを、非 volatile 修飾型の左辺値を使って変更しようとした場合、メッセージを出力します。

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

(29) 関数定義
可変個引数の実引数を受け付ける関数が、省略記号表記で終わる仮引数型並びをもたずに定義された場合、仮引数の値が不定となります。

(30) 条件付取り込み
置き換え処理によって字句 defined が生成される場合、または defined 単項演算子のマクロ置き換え前の使用法が制約の中で規定した二つの形式のどちらにも一致しない場合、通常の defined として扱います。

(31) マクロ置き換えー前処理句を含まない実引数
実引数が（実引数の置換前に）前処理句を含まない場合、メッセージを出力します。

(32) マクロ置き換えー前処理指令を持つ実引数
実引数の並びの中に、ほかの場合であれば前処理指令として働く前処理字句列がある場合、メッセージを出力します。

(33) # 演算子
置き換えの結果が、正しい単純文字列リテラルでない場合、メッセージを出力します。

(34) ## 演算子
置き換えの結果が、正しい単純文字列リテラルでない場合、メッセージを出力します。

4.1.3 C90 の処理系定義

(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)
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)
先頭から 8189 文字までを意味のあるものとして扱います。

(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)
先頭から 8191 文字までを意味のあるものとして扱います。

(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)
シフト状態に依存した表現形式を持ちません。

(9) 実行文字集合の文字におけるビット数。(5.2.4.2.1)
文字は 8 ビットです。多バイト文字は 16 ビットです。

(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)
ソース文字集合の要素と、実行文字集合の要素は一致します。

(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)
4 バイト（ASCII であれば 4 文字相当）の文字を含む整数文字定数の値は、4 バイト全てが有効値となります。
5 バイト以上はエラーメッセージを出力します。

(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)
「4.1.5 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)
少ないビット数へ変換する場合は、少ないビット数のビット幅でマスクをした（上位ビットを削除した）ビット列とします。符号なし整数を同じビット数の符号付き整数に変換する場合は、ビット列をそのままコピーします。

(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)
オペランドが負の値をもつ場合、"%" 演算子の結果の符号は第 1 オペランド（被除数）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)
“E1 >> E2” において、E1 が符号付きの型で負の値をもつ場合、算術シフトを行います。

(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)
「4.1.5 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)
次のいずれかの条件を満たす場合、最近値の方向に切り捨てます。
-nofpu を指定
-cpu=rx200 を指定
-cpu=rx600 または -isa=rxv1 を指定し、かつ変換する対象の整数が符号なしである
それ以外の場合、単精度浮動小数点型への変換結果は FPSW の RM[0:1] ビットに従います。
【V3.01.00 以降】-dpfpu を指定した場合、倍精度浮動小数点型への変換結果は DPSW の DRM[0:1] ビットに従います。

(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)
最近値に丸めます。

(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 long 型とします。

(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)

- 整数からポインタへの変換結果
  整数型のサイズがポインタ型のサイズより大きい場合、または整数型のサイズとポインタ型のサイズが等しい場合は、整数型の下位バイトの値になります。整数型のサイズがポインタ型のサイズより小さい場合は、符号拡張した値になります。
- ポインタから整数への変換結果
  ポインタ型のサイズが整数型のサイズより大きい場合、またはポインタ型のサイズと整数型のサイズが等しい場合は、ポインタ型の下位バイトの値になります。ポインタ型のサイズが整数型のサイズより小さい場合は、ゼロ拡張した値になります。

(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)
long 型とします。

(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)
記憶域クラス指定子 “register” の宣言の有無にかかわらず、可能なかぎり高速にアクセスするように最適化を行います。

(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)
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)
「4.1.5 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)
符号を明示せずに宣言されたビットフィールドは、unsigned int 型として扱います。

(30) 単位内のビットフィールドの割付け順序。(6.5.2.1)
最初に宣言されたビットフィールドは、ビットフィールド宣言時の型のサイズの領域の最下位ビットから割り当てられます。ただし、オプションにより変更可能です。
「4.1.5 データの内部表現と領域」を参照してください。

(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)
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。
「4.1.5 データの内部表現と領域」を参照してください。

(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)
signed long 型です。オプション -auto_enum を指定した場合、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。アクセス幅は宣言型より小さなサイズでアクセスすることがあります。

(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)
制限はありません。

(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一致します。
負の値を持ちません。

(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
フルパス指定がない場合は
2. ソース・ファイルがあるフォルダ
3. オプション -include 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（環境変数 INC_RX 指定フォルダ）

(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
フルパス指定がない場合は
2. ソース・ファイルがあるフォルダ
3. オプション -include 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（環境変数 INC_RX 指定フォルダ）

(39) ソース・ファイル名と文字列との対応付け。(6.8.2)
< と >、"" で囲まれた前処理字句列は、そのままヘッダ名に対応付けられます。マクロが展開され < 文字列 >、" 文字列 " の形式になった場合は、その内容をヘッダ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)
「4.2.3 #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)
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)
コンパイル時の lang オプションにより表示は変化します。
-lang=c99 がないとき (C(C89)、C++、EC++ 言語の場合) :
ASSERTION FAILED:Δ 式 ΔFILEΔ< ファイル名 >,LINEΔ< 行番号 >
-lang=c99 があるとき (C(C99) 言語の場合) :
ASSERTION FAILED:Δ 式 ΔFILEΔ< ファイル名 >,LINEΔ< 行番号 >ΔFUNCNAMEΔ< 関数名 >
assert 関数終了時の動作は規定しません。低水準インタフェースルーチンの仕様によります。

(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)
非数を返します。
「4.1.5 データの内部表現と領域」の「(5) 浮動小数点型の仕様」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)
アンダフロー発生時に errno に ERANGE を設定する関数については、「10.5.6 C 標準ライブラリ関数のエラーメッセージ」を参照してください。それ以外は設定しません。

(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)
signal 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)
signal 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)
signal 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)
signal 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)
signal 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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
規定しません。低水準インタフェースルーチンの仕様によります。

(58) ファイルバッファリングの特性。(7.9.3)
規定しません。低水準インタフェースルーチンの仕様によります。

(59) 長さ 0 のファイルが実際に存在するか否か。(7.9.3)
規定しません。低水準インタフェースルーチンの仕様によります。

(60) 正しいファイル名の規則。(7.9.3)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
remove 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)
rename 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 進数出力です。

(65) fscanf 関数中の %p 変換に対する入力。(7.9.6.2)
16 進数入力です。

(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)
「^」の直後でない場合、直前の文字と直後の範囲を示します

(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)
fgetpos 関数はサポートしていません。
ftell 関数については規定しません。低水準インタフェースルーチンの仕様によります。

(68) perror 関数によって生成されるメッセージ。(7.9.10.4)
「7.4 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)
規定しません。低水準インタフェースルーチンの仕様によります。

(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)
exit 関数はサポートしていません。

(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)
「10.5.6 C 標準ライブラリ関数のエラーメッセージ」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)
clock 関数はサポートしていません。

4.1.4 C99 の処理系定義

(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終了処理ルーチンを作成してください。

(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います。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）
オプション指定による、入力プログラムの文字コード指定と出力ファイルの文字コード指定が等しい場合は、同一の値をもつ要素へ対応付けます。オプション指定が異なる場合は、それぞれ対応する文字コードの値となります。

(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）
4 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 4 バイトの値を持ちます。5 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 の補数を使用して表現されるかどうか、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）
下記いずれかの条件を満たす場合は最も近い方向に丸められます。
-nofpu を指定
-cpu=rx200 を指定
-cpu=rx600 または -isa=rxv1 を指定し、かつ変換対象の整数が符号なし
それ以外の場合は FPSW の RM[0:1] ビットに従います。
【V3.01.00 以降】-dpfpu を指定した場合、倍精度浮動小数点型への変換結果は DPSW の DRM[0:1] ビットに従います。

(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）
オプション -round の指定、およびマイコンの設定に従います。

(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）
オプション -round の指定に従います。

(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、fenv.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）
nearbyint 関数、nearbyintf 関数、nearbyintl 関数は " 不正確結果 " 浮動小数点例外を生成しません。rint 関数、rintf 関数、rintl 関数は " 不正確結果 " 浮動小数点例外を生成する場合があります。

(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）
オプション -round の指定、およびマイコンの設定に従います。

(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）
- 整数からポインタへの変換結果
整数型のサイズがポインタ型のサイズより大きい場合は、整数型の下位バイトの値になります。整数型のサイズとポインタ型のサイズが等しい場合は、整数型のビットパターンがそのまま保持されます。整数型のサイズがポインタ型のサイズより小さい場合は、long 型に拡張した結果の値をそのまま保持します。
- ポインタから整数への変換結果
ポインタ型のサイズが整数型のサイズより大きい場合は、ポインタ型の下位バイトの値になります。ポインタ型のサイズと整数型のサイズが等しい場合は、ポインタ型のビットパターンがそのまま保持されます。ポインタ型のサイズが整数型のサイズより小さい場合は、ポインタ型の値をゼロ拡張した値になります。

(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 long 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）
オプション -inline が有効な場合にだけ展開を試みます。また、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。オプション -bit_order または #pragma bit_order で選択が可能です。

(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）
「4.1.5 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）
signed long 型です。ただし、オプション -auto_enum 指定時は、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。アクセス幅は宣言型より小さなサイズでアクセスすることがあります。

(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 -include 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 -include 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）
「4.2.3 #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* FILE *ファイル名* ,LINE *行番号* FUNCNAME *関数名*

(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）
下記のいずれか、または組み合わせになります。

_FE_DIVBYZERO	0x04
_FE_INEXACT	0x10
_FE_INVALID	0x01
_FE_OVERFLOW	0x02
_FE_UNDERFLOW	0x08

(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）
" オーバーフロー " 浮動小数点例外生成時に FPU の E0 フラグが 0 の場合、" 不正確結果 " 浮動小数点例外を生成する可能性があります。それ以外では " 不正確結果 " 浮動小数点例外を生成しません。

(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）
(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）

acosh 関数群	引数を x とします。x<1 が成り立つとき、定義域エラーとなります。このときの戻り値は NaN です。
atanh 関数群	引数を x とします。-1<x<1 が成り立つとき、定義域エラーとなります。このときの戻り値は NaN です。
ccosh 関数群	引数を x とします。 ・x の実部が 0 であり , かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN+0×i です。 ・x の実部が ±∞ であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は ∞+NaN×i です。 ・x の実部が有限値であり、かつ x の虚部が ±∞ であるとき , 定義域エラーとなります。このときの戻り値は NaN+NaN×i です。
cexp 関数群	引数を x とします。 ・x の実部が +∞ であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は +∞+NaN×i です。 ・x の実部が有限値であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN+NaN×i です。
csinh 関数群	引数を x とします。 ・x の実部が 0 であるか、または ±∞ であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は x の実部 +NaN×i です。 ・x の実部有限値であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN+NaN×i です。
fma 関数群	第 1 引数を x1, 第 2 引数を x2, 第 3 引数を x3 とします。 ・x1 が ±∞ かつ x2 が 0 かつ x3 が NaN 以外であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が 0 かつ x2 が ±∞ かつ x3 が NaN 以外であるとき、定義域エラーとなります (規定なし)。このときの戻り値は NaN です。 ・x1, x2 及び x3 がいずれも ±∞ であり、かつ x1×x2 と x3 の符号が異なる (すなわち、∞ 同士の減算が生じる) とき、定義域エラーとなります。このときの戻り値は NaN です。
llrint 関数群	引数を x とします。 ・x が有限値であり、かつ longlong 型で表せない値であるとき、定義域エラーとなります。このときの戻り値は 0 です。 ・x が NaN であるか、または ±∞ であるとき、定義域エラーとなります。このときの戻り値は 0 です。
llround 関数群	引数を x とします。 ・x が有限値であり、かつ longlong 型で表せない値であるとき、定義域エラーとなります。このときの戻り値は 0 です。 ・x が NaN であるか、または ±∞ であるとき、定義域エラーとなります。このときの戻り値は 0 です。
log1p 関数群	引数を x とします。x が x<-1 を満たすとき、定義域エラーとなります。このときの戻り値は NaN です。
lrint 関数群	引数を x とします。 ・x が有限値であり、かつ long 型で表せない値であるとき、定義域エラーとなります。このときの戻り値は 0 です。 ・x が NaN であるか、または ±∞ であるとき、定義域エラーとなります。このときの戻り値は 0 です。
lround 関数群	引数を x とします。 ・x が有限値であり、かつ long 型で表せない値であるとき、定義域エラーとなります。このときの戻り値は 0 です。 ・x が NaN であるか、または ±∞ であるとき、定義域エラーとなります。このときの戻り値は 0 です。

remquo 関数群	第 1 引数を x1, 第 2 引数を x2, 第 3 引数を x3 とします。 ・x1 が ±∞ であり、かつ x2 が NaN ではないとき、定義域エラーとなります。このときの戻り値は NaN です。なお、x3 の指し先は 0 になります。 ・x1 が NaN ではなく、かつ x2 が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。なお、x3 の指し先は 0 になります。
tgamma 関数群	引数を x とします。x が -∞ であるか、または負の整数であるとき、定義域エラーとなります。このときの戻り値は NaN です。
carg 関数群	引数を x とします。x の実部が 0 であり、かつ x の虚部が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。
ccos 関数群	引数を x とします。 ・x の実部が ±∞ であり、かつ x の虚部が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN+0×i です。 ・x の実部が ±∞ であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は ∞+NaN×i です。 ・x の実部が ±∞ であり、かつ x の虚部が有限値であるとき、定義域エラーとなります。このときの戻り値は NaN+NaN×i です。
clog 関数群	引数を x とします。x の実部が ±∞ であり、かつ x の虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は ±∞+ NaN×i です。
cpow 関数群	第 1 引数を x1, 第 2 引数を x2 とします。 ・x1 の虚部が 0 であり、かつ x2 を x1 の実部倍した結果の実部が +∞ であり、かつ虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は +∞+NaN×i です。 ・x1 の虚部が 0 であり、かつ x2 を x1 の実部倍した結果の実部が有限値であり、虚部が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN+NaN×i です。 ・x1 または x2 の実部が NaN であり、かつ x1 及び x2 の虚部がいずれも 0 であるとき、定義域エラーとなります。このときの戻り値は NaN+0×i です。 ・x1 の実部が 0 であり、かつ x2 の実部が 0 以下であり、かつ x1 及び x2 の虚部がいずれも 0 であるとき、定義域エラーとなります。このときの戻り値は NaN+0×i です。 ・x1 の実部が負の数であり、かつ x2 の実部が非整数であり、かつ x1 及び x2 の虚部がいずれも 0 であるとき、定義域エラーとなります。このときの戻り値は NaN+0×i です。
remainder 関数群	第 1 引数を x1, 第 2 引数を x2 とします。 ・x1 が ±∞ であり、かつ x2 が NaN ではないとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が NaN ではなく、かつ x2 が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。
acos 関数群	引数を x とします。 ・x が x<-1 または 1<x を満たすとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
asin 関数群	引数を x とします。 ・x が x<-1 または 1<x を満たすとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
atan 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
cosh 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。

exp 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
frexp 関数群	第 1 引数を x1, 第 2 引数を x2 とします。x1 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。なお、x2 の指し先は変更しません。
log10 関数群	引数を x とします。 ・x<0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
log 関数群	引数を x とします。 ・x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x<0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。
sin 関数群	引数を x とします。x が ±∞ であるか、または NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
cos 関数群	引数を x とします。x が ±∞ であるか、または NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
sinh 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
tan 関数群	引数を x とします。x が ±∞ であるか、または NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
tanh 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
atan2 関数群	第 1 引数を x1, 第 2 引数を x2 とします。 ・x1 が 0 であり、かつ x2 が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 または x2 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が ±∞ であり、かつ x2 が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN です。
ceil 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
floor 関数群	引数を x とします。x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
fmod 関数群	第 1 引数を x1, 第 2 引数を x2 とします。 ・x2 が 0 であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 または x2 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が ±∞ であるとき、定義域エラーとなります。このときの戻り値は NaN です。
ldexp 関数群	第 1 引数を x1, 第 2 引数を x2 とします。x1 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。
modf 関数群	第 1 引数を x1, 第 2 引数を x2 とします。x1 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。なお、x2 の指し先は NaN になります。

pow 関数群	第 1 引数を x1, 第 2 引数を x2 とします。 ・x1 が NaN であるか、または x2 が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が 0 であり、かつ x2 が 0 以下であるとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x1 が x1<0 を満たし、かつ x2 が整数ではないとき、定義域エラーとなります。このときの戻り値は NaN です。
sqrt 関数群	引数を x とします。 ・x が x<0 を満たすとき、定義域エラーとなります。このときの戻り値は NaN です。 ・x が NaN であるとき、定義域エラーとなります。このときの戻り値は NaN です。

(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 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を生成します。

(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）
31 とします。

(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）
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）
低水準インタフェースルーチンの仕様に依存します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(83) ファイルバッファリングの特性。（7.19.3）
低水準インタフェースルーチンの仕様に依存します

(84) 長さ 0 のファイルが実際に存在するかどうか。（7.19.3）
低水準インタフェースルーチンの仕様に依存します。

(85) 正しいファイル名の規則。（7.19.3）
低水準インタフェースルーチンの仕様に依存します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(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）
多バイト文字の表現形式としてシフト状態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）
remove 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）
rename 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）
tmpfile 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）
ファイル名がない場合は現在のストリームを指定したモードに変更します。

(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 +++++++、負の無限大は -------、NaN は ******* 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。

(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）
直前の文字と直後の文字との範囲を示します。

(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。
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）
ftell 関数は低水準インタフェースルーチンの仕様に依存します。
fgetpos 関数、fsetpos 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）
strtof 関数、strtold 関数、wcstod 関数、wcstof 関数、wcstold 関数は qNaN と解釈します。strtod 関数は浮動小数点型の数値以外と解釈します。

(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、wcstod 関数、wcstof 関数、wcstold 関数はグローバル変数 errno に ERANGE を設定します。

(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）
低水準インタフェースルーチンの仕様に依存します。

(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）
abort 関数、exit 関数、_Exit 関数はサポートしていません。

(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）
hypot 関数群、ldexp 関数群、lgamma 関数群、tgamma 関数群、erfc 関数群、pow 関数群、scalbln 関数群、tan 関数群、exp 関数群、nexttoward 関数群で結果がオーバーフローまたはアンダーフローした場合に " 不正確結果 " 浮動小数点例外を生成する可能性があります。

(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）
lround 関数群は丸め方向モードに従わないケースがあります。

(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）
「7. ライブラリ関数仕様」にある各ヘッダ・ファイル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）
「4.1.5 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）
「4.1.5 データの内部表現と領域」を参照してください。

翻訳限界
C99 規格の翻訳限界に対する CC-RX の仕様を示します。
「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。

項目	限界値
ブロックの入れ子のレベル数	制限なし
条件付き取込みにおける入れ子のレベル数	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	128
一つの完結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	制限なし
一つの完結式における括弧で囲まれた式の入れ子のレベル数	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	制限なし
外部識別子において意味がある先頭の文字数	制限なし
一つの翻訳単位中における外部識別子数	制限なし
一つのブロックで宣言されるブロック有効範囲をもつ識別子数	制限なし
一つの前処理翻訳単位中で同時に定義されるマクロ識別子数	制限なし
一つの関数定義における仮引数の個数	制限なし
一つの関数呼出しにおける実引数の個数	制限なし
一つのマクロ定義における仮引数の個数	制限なし
一つのマクロ呼出しにおける実引数の個数	制限なし
一つの論理ソース行における文字数	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	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	2147483647
#include で取り込まれるファイルの入れ子のレベル数	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 ラベルの個数	2147483647
一つの構造体又は共用体のメンバ数	制限なし
一つの列挙体における列挙定数の個数	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	制限なし

4.1.5 データの内部表現と領域

本節では、型名と、データの内部表現の対応について述べます。データの内部表現は、以下の項目から成り立っています。

- データのサイズ
  データの占有する領域のサイズです。
- データのアライメント数
  データを割り付けるアドレスに関する制約です。任意のアドレスに割り付ける 1 バイトアライメント、偶数バイトに割り付ける 2 バイトアライメント、4 の倍数バイトに割り付ける 4 バイトアライメントがあります。
- 値の範囲
  スカラ型 (C 言語)、基本型 (C++ 言語) の値のとり得る範囲を示します。
- データの割り付け例
  構造体 / 共用体 (C 言語)、クラス型 (C++ 言語) の要素となるデータの割り付け方を示します。

(1) スカラ型 (C 言語)、基本型 (C++ 言語)
C 言語におけるスカラ型および、C++ 言語における基本型の内部表現を表 4.1 に示します。

表 4.1 スカラ型・基本型の内部表現

	型名	サイズ (byte)	アライメント数 (byte)	符号の有無	値の範囲	
					最小値	最大値
1	char *1	1	1	無	0	2^8-1 (255)
2	signed char	1	1	有	-2^7 (-128)	2^7-1 (127)
3	unsigned char	1	1	無	0	2^8-1 (255)
4	short	2	2	有	-2^{15} (-32768)	2^{15}-1 (32767)
5	signed short	2	2	有	-2^{15} (-32768)	2^{15}-1 (32767)
6	unsigned short	2	2	無	0	2^{16}-1 (65535)
7	int *2	4	4	有	-2^{31}(-2147483648)	2^{31}-1 (2147483647)
8	signed int *2	4	4	有	-2^{31}(-2147483648)	2^{31}-1 (2147483647)
9	unsigned int *2	4	4	無	0	2^{32}-1 (4294967295)
10	long	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
11	signed long	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
12	unsigned long	4	4	無	0	2^{32}-1 (4294967295)
13	long long	8	4	有	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
14	signed long long	8	4	有	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
15	unsigned long long	8	4	無	0	2^{64}-1 (18446744073709551615)
16	float	4	4	有	-∞	+∞

	型名	サイズ (byte)	アライメント数 (byte)	符号の有無	値の範囲	
					最小値	最大値
17	double long double	4 *4	4	有	-∞	+∞
18	size_t	4	4	無	0	2^{32}-1 (4294967295)
19	ptrdiff_t	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
20	enum *3	4	4	有	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
21	ポインタ	4	4	無	0	2^{32}-1 (4294967295)
22	bool *5 _Bool *8	1 *9	1 *9	-*9	-	-
23	リファレンス *6	4	4	無	0	2^{32}-1 (4294967295)
24	データメンバへのポインタ *6	4	4	無	0	2^{32}-1 (4294967295)
25	関数メンバへのポインタ *6 *7	12	4	- *10	-	-

注 1. signed_char オプションを指定した場合、signed char 型と同じ値の範囲を持ちます。

注 2. int_to_short オプションを指定した場合、int 型は short 型と、signed int 型は signed short 型と、unsigned int 型は unsigned short 型とそれぞれ同じ値の範囲を持ちます。

注 3. auto_enum オプションを指定した場合、列挙値が収まる最小の型となります。

注 4. dbl_size=8 を指定した場合、double 型および long double 型のサイズは 8 バイトになります。

注 5. C++ プログラム、または stdbool.h をインクルードした C プログラムのコンパイル時のみ有効です。

注 6. C++ プログラムのコンパイル時のみ有効です。

注 7. 関数メンバ・仮想関数メンバへのポインタは、以下のデータ構造で表現しています。

```
class PMF{
 public:
   long d;            // オブジェクトのオフセット値
   long i;            // 対象メンバ関数が仮想関数のときの仮想関数表中での
                        // インデックス
   union{
     void (*f)();    // 対象メンバ関数が非仮想関数のときの関数のアドレス
     long offset;    // 対象メンバ関数が仮想関数のときの仮想関数表のオブジェクト
                     // 中のオフセット
   };
};
```

注 8. C99 コンパイル、または stdbool.h をインクルードした C プログラムのコンパイル時に有効です。

注 9. C89 でコンパイルする場合は、unsigned long 型と同じサイズ、アライメント数及び符号になります。

注 10. 符号の設定はありません。

(2) 構造体 / 共用体 (C 言語)、クラス型 (C++ 言語)

本項では、C 言語における配列型、構造体型、共用体型および、C++ 言語におけるクラス型の内部表現について説明します。

表 4.2 に構造体 / 共用体、クラス型の内部表現を示します。

表 4.2 構造体 / 共用体、クラス型の内部表現

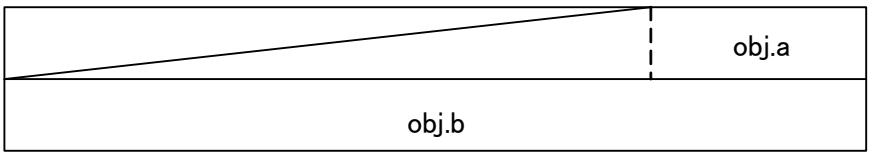
	型名	アライメント数 (byte)	サイズ (byte)	データの割り付け例
1	配列型	配列要素のアライメント数	配列要素の数 × 要素サイズ	char a[10]; アライメント数 1byte サイズ 10byte
2	構造体型	構造体メンバのアライメント数のうち最大値	メンバのサイズの和 「(a) 構造体データの割り付け方」参照	struct { char a,b; }; アライメント数 1byte サイズ 2byte
3	共用体型	共用体メンバのアライメント数のうち最大値	最大メンバのサイズ 「(b) 共用体データの割り付け方」参照	union { char a,b; }; アライメント数 1byte サイズ 1byte
4	クラス型	仮想関数がある場合 : 常に 4 2) 上記以外 : データメンバのアライメント数のうち最大値	データメンバ、仮想関数表へのポインタ、仮想基底クラスへのポインタの和 「(c) クラスデータの割り付け方」参照	class B: public A{ virtual void f(); }; アライメント数 4byte サイズ 8byte class A{ char a; }; アライメント数 1byte サイズ 1byte

以下の例でサイズを明記していない □ は、4 バイトを表します。／ はパディングを表します。
また、アドレスの増える向きは、右から左とします（左側が上位アドレス）。

(a) 構造体データの割り付け方
構造体型の各メンバを割り付ける場合、そのメンバの型名の境界調整数に合わせるために直前のメンバとの間にパディングが生じる場合があります。

例

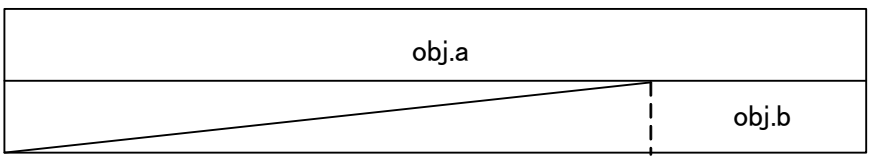
```
struct {
  char a;
  int b;
} obj;
```



構造体が 4 バイトのアライメント数を持ち、最後のメンバが 1,2,3 バイト目で終わっている場合、その次のバイトも含めて構造体型の領域として扱います。

例

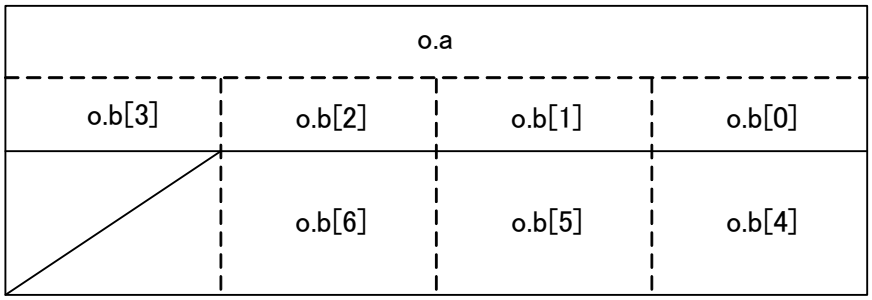
```
struct {
  int a;
  char b;
} obj;
```



(b) 共用体データの割り付け方
共用体が 4 バイトのアライメント数を持ち、最大メンバのサイズが 4 の倍数バイトでない場合、4 の倍数になるまで残りのバイトも含めて共用体型の領域として扱います。

例

```
union {
  int a;
  char b[7];
} o;
```

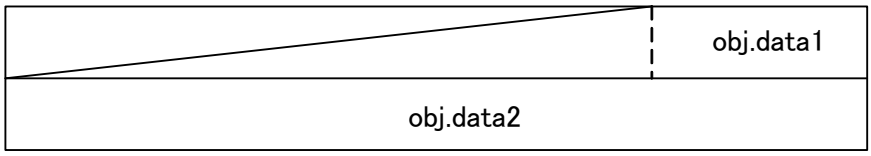


(c) クラスデータの割り付け方

基底クラス、仮想関数がないクラスの場合、構造体データの割り付け規則に従ってデータメンバを割り付けます。

例

```
class A{
   char data1;
   int data2;
 public:
   A();
   char getData1(){return data1;}
}obj;
```



アライメント数が 1 の基底クラスから派生したクラスの先頭メンバが 1byte データの場合、パディングを作らないようにデータメンバを割り付けます。

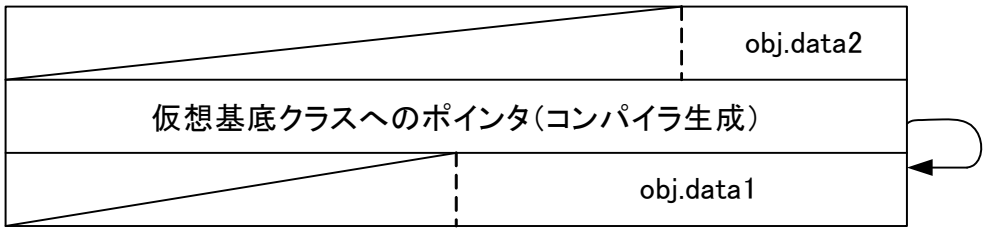
例

```
class A{
   char data1;
};
class B:public A{
   char data2;
   short data3;
}obj;
```

クラスに仮想基底クラスがある場合、仮想基底クラスへのポインタを割り付けます。

例

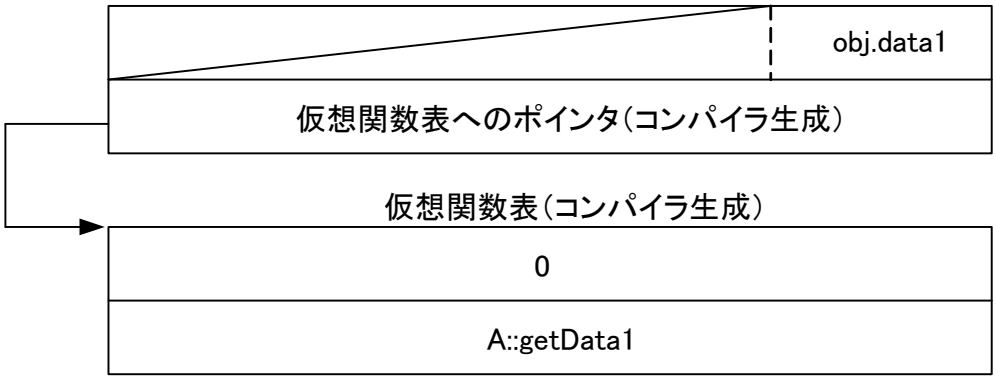
```
class A{
   short data1;
};
class B: virtual protected A{
   char data2;
}obj;
```



クラスに仮想関数がある場合、コンパイラは仮想関数表を生成し、仮想関数表へのポインタを割り付けます。

例

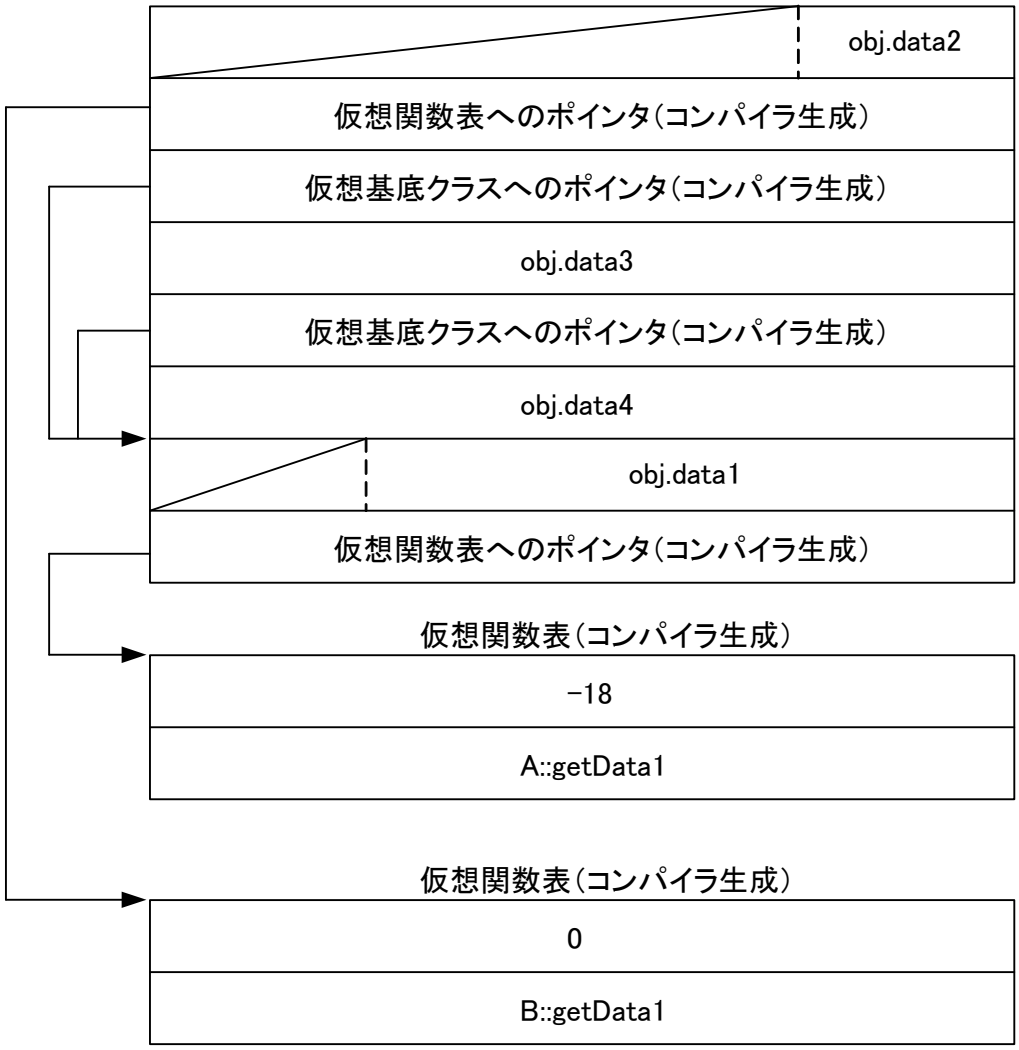
```
class A{
   char data1;
 public:
   virtual char getData1();
}obj;
```



仮想基底クラス、基底クラス、仮想関数があるクラスの例を示します。

例

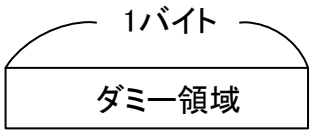
```
class A{
   char data1 ;
   virtual char getData1();
};
class B:virtual public A{
   char data2;
   char getData2();
   char getData1();
};
class C:virtual protected A{
   int data3;
};
class D:virtual public A,public B,public C{
 public:
   int data4;
   char getData1();
}obj;
```



空クラスの場合、1バイトのダミー領域を割り付けます。

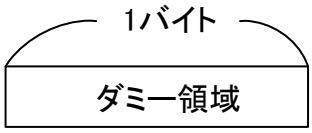
例

```
class A{
   void fun();
}obj;
```



空クラスを基底クラスに持つ空クラスの場合でも、ダミー領域は1バイトになります。

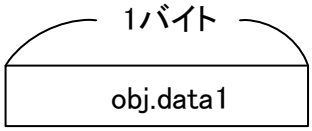
```
class A{
   void fun();
};
class B: A{
   void sub();
}obj;
```



空クラスのダミー領域は、クラスサイズが 0 の場合に割り付けます。基底クラスや派生クラスにデータメンバがある場合や、仮想関数があるクラスの場合には、ダミー領域は割り付けません。

例

```
class A{
   void fun();
};
class B: A{
   char data1;
}obj;
```



(3) ビットフィールド

ビットフィールドは、構造体、共用体、クラスの中にビット幅を指定して割り付けるメンバです。
本項では、ビットフィールド特有の割り付け規則について説明します。

(a) ビットフィールドのメンバ

表 4.3 にビットフィールドメンバの仕様を示します。

表 4.3 ビットフィールドメンバの仕様

	項目	仕様
1	ビットフィールドで許される型指定子	(unsigned) char、signed char、bool*1、_Bool*5、 (unsigned) short、signed short、enum、 (unsigned) int、signed int、 (unsigned) long、signed long、 (unsigned) long long、signed long long
2	宣言された型に拡張するときの符号の扱い *2	符号なし (unsigned) は、ゼロ拡張 *3 符号付き (signed) は、符号拡張 *4
3	符号指定なしの型の符号型	符号なし (unsigned) 但し、signed_bitfield オプションが指定された場合は、符号付き (signed)
4	enum 型の符号型	符号付き (signed) 但し、auto_enum オプションが指定された場合は、その結果の型に従う

注 1. C++ プログラム、または stdbool.h をインクルードした C プログラムのコンパイル時のみ bool を指定できます。

注 2. ビットフィールドのメンバを使用する場合、ビットフィールドに格納したデータを宣言した型に拡張して使用します。符号付き (signed) で宣言されたサイズが 1 ビットのビットフィールドのデータは、データそのものを符号として解釈します。したがって、表現できる値は 0 と -1 だけになります。

注 3. ゼロ拡張：拡張するときに上位のビットにゼロを補います。

注 4. 符号拡張：拡張するときにビットフィールドデータの最上位ビットを符号として解釈し、データより上位のビット全てに符号ビットを補います。

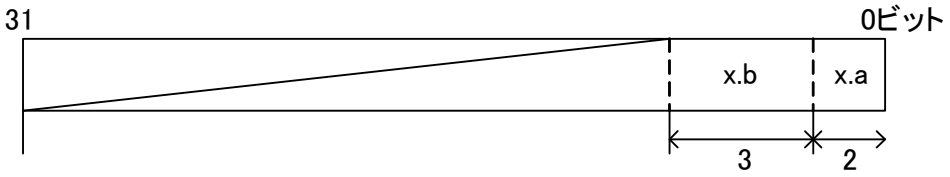
注 5. C99 プログラムのみ有効です。_Bool 型は bool 型と同じ型としてコンパイルされます。

(b) ビットフィールドの割り付け方
ビットフィールドは、以下の５つの規則に従って割り付けます。

- ビットフィールドのメンバは領域内で右 (下位ビット側) から順に詰め込みます。

例

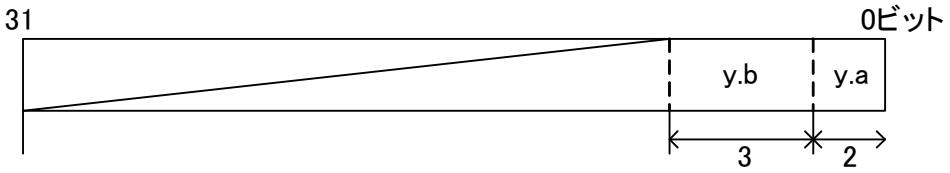
```
struct b1 {
  int a:2;
  int b:3;
} x;
```



- 同じサイズの型指定子が連続している場合は、可能な限り同じ領域に詰め込みます。

例

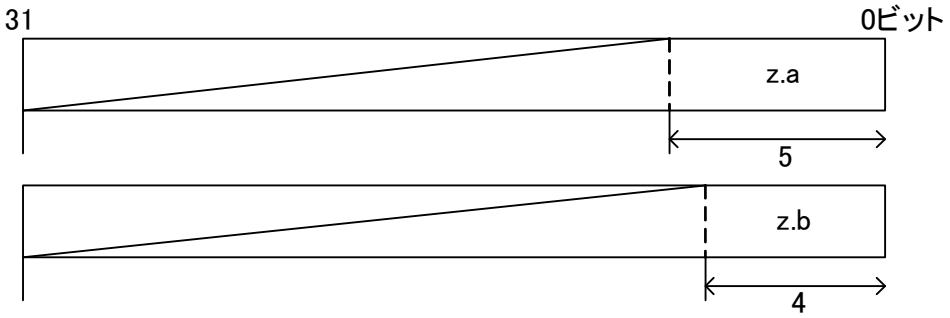
```
struct b1 {
  long          a:2;
  unsigned int  b:3;
} y;
```



異なるサイズの型指定子で宣言されたメンバは、次の領域に割り付けます。

例

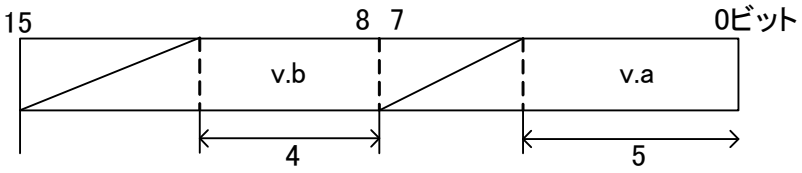
```
struct b1 {
  int   a:5;
  char  b:4;
} z;
```



- 同じサイズの型指定子が連続していても、詰め込み先の領域の残りビットが、次のビットフィールドのサイズより小さい場合は、残りの領域は未使用領域となり、次の領域に割り付けます。

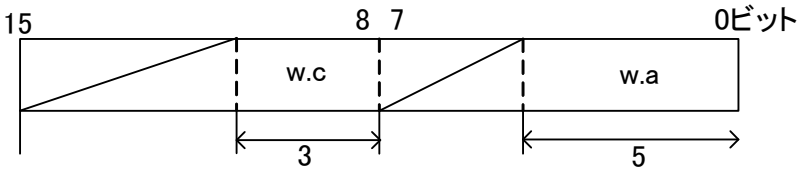
例

```
struct b2 {
  char a:5;
  char b:4;
} v;
```



- ビット幅 0 のビットフィールドのメンバを指定すると、次のメンバからは、強制的に次の領域に割り付けます。

```
struct b2 {
  char a:5;
  char  :0;
  char c:3;
} w;
```



備考 ビットフィールドメンバを上位ビット側から詰め込むことも可能です。
詳細は、「4.2 拡張言語仕様」の #pragma bit_order および「オプション」の章の bit_order オプションを参照してください。

(4) Big Endian のメモリ割り付け
Big Endian でのメモリ上のデータ配列は以下のとおりです。

(a) 1 バイトデータ (char、signed char、unsigned char、bool[*1]、_Bool[*1] 型)
1 バイトデータの中のビット並び順は、Little Endian の場合も、Big Endian の場合も同じです。

注 1. C89 でコンパイルする場合は、サイズ、アライメント数共に 4 になります。

(b) 2 バイトデータ ((signed)short、unsigned short 型)
2 バイトデータの中のバイト並び順は、Little Endian と Big Endian で上位、下位のバイトが逆になります。

例
0x100 番地に 2 バイトデータ 0x1234 がある場合

Little Endian:	0x100 番地：0x34	Big Endian:	0x100 番地：0x12
	0x101 番地：0x12		0x101 番地：0x34

(c) 4 バイトデータ ((signed)int[*2]、unsigned int[*2]、(signed)long、unsigned long、float 型)
4 バイトデータの中のバイト並び順は、Little Endian と Big Endian で 4 バイトのデータの順序が逆になります。

注 2. int_to_short オプションを指定した場合、signed int 型は signed short 型と、unsigned int 型は unsigned short 型とそれぞれ同じサイズ及びアライメント数になります。

例
0x100 番地に 4 バイトデータ 0x12345678 がある場合

Little Endian:	0x100 番地：0x78	Big Endian:	0x100 番地：0x12
	0x101 番地：0x56		0x101 番地：0x34
	0x102 番地：0x34		0x102 番地：0x56
	0x103 番地：0x12		0x103 番地：0x78

(d) 8 バイトデータ **(**(signed)long long、unsigned long long、double 型)
8 バイトデータの中のバイト並び順は、Little Endian と Big Endian で 8 バイトのデータの順序が逆になります。

例
0x100 番地に 8 バイトデータ 0x0123456789abcdef がある場合

Little Endian:	0x100 番地：0xef	Big Endian:	0x100 番地：0x01
	0x101 番地：0xcd		0x101 番地：0x23
	0x102 番地：0xab		0x102 番地：0x45
	0x103 番地：0x89		0x103 番地：0x67
	0x104 番地：0x67		0x104 番地：0x89
	0x105 番地：0x45		0x105 番地：0xab
	0x106 番地：0x23		0x106 番地：0xcd
	0x107 番地：0x01		0x107 番地：0xef

(e) 構造体 **/** 共用体、クラス型データ
構造体 / 共用体、クラス型データの各メンバの割り付けは Little Endian のときと同様です。ただし、各メンバのバイト並び順はそのデータサイズの規則に従って反転します。

例
0x100 番地に、

```
struct {
    short a;
    int b;
}z = {0x1234, 0x56789abc};
```

がある場合

Little Endian:	0x100 番地：0x34	Big Endian:	0x100 番地：0x12
	0x101 番地：0x12		0x101 番地：0x34
	0x102 番地：パディング		0x102 番地：パディング
	0x103 番地：パディング		0x103 番地：パディング
	0x104 番地：0xbc		0x104 番地：0x56
	0x105 番地：0x9a		0x105 番地：0x78
	0x106 番地：0x78		0x106 番地：0x9a
	0x107 番地：0x56		0x107 番地：0xbc

(f) ビットフィールド
ビットフィールドの各領域の割り付けも Little Endian のときと同様です。ただし、各領域のバイト並び順はそのデータサイズの規則に従って反転します。

例
0x100 番地に、

```
struct {
    long a:16;
    unsigned int b:15;
    short c:5;
}y={1,1,1};
```

がある場合

Little Endian:	0x100 番地：0x01	Big Endian:	0x100 番地：0x00
	0x101 番地：0x00		0x101 番地：0x01
	0x102 番地：0x01		0x102 番地：0x00
	0x103 番地：0x00		0x103 番地：0x01
	0x104 番地：0x01		0x104 番地：0x00
	0x105 番地：0x00		0x105 番地：0x01
	0x106 番地：パディング		0x106 番地：パディング
	0x107 番地：パディング		0x107 番地：パディング

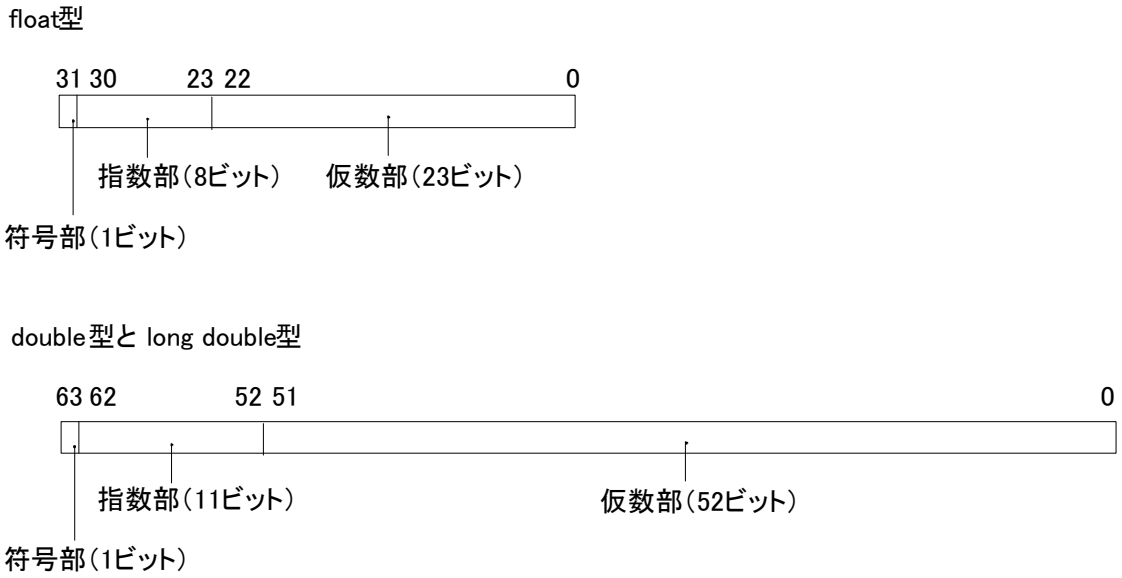
(5) 浮動小数点型の仕様

(a) 浮動小数点型の内部表現
コンパイラで扱う浮動小数点型の内部表現は、IEEE の形式に準拠しています。ここでは、IEEE 形式の浮動小数点型の内部表現の概要について述べます。
なお、本節では dbl_size=8 オプションが指定されたものとして説明しています。dbl_size=4 オプションが指定された場合は、double 型および long double 型の内部表現は、float 型と同じになります。

(b) 内部表現の形式
float 型は IEEE の単精度形式 (32 ビット)、double 型と long double 型は IEEE の倍精度形式 (64 ビット) で表現します。

(c) 浮動小数点データフォーマット
float 型および double 型と long double 型の浮動小数点データフォーマットを図 4.1 に示します。

図 4.1 浮動小数点データフォーマット



内部表現の各構成要素の意味を以下に示します。
(i) 符号部
浮動小数点型の符号を示します。0 のとき正、1 のとき負を示します。
(ii) 指数部
浮動小数点型の指数を 2 のべき乗で示します。
(iii) 仮数部
浮動小数点型の有効数字に対応するデータです。

(d) 表現する値の種類
浮動小数点型は、通常の実数値のほかに、無限大等の値も表現することができます。浮動小数点型が表現する値の種類を以下に示します。
(i) 正規化数
指数部が 0 または全ビット 1 ではない場合です。通常の実数値を表現します。
(ii) 非正規化数
指数部が 0 で、仮数部が 0 でない場合です。絶対値の小さな実数値を表現します。
(iii) ゼロ
指数部および仮数部が 0 の場合です。値 0.0 を表現します。
(iv) 無限大
指数部が全ビット 1 で仮数部が 0 の場合です。無限大を表現します。
(v) 非数
指数部が全ビット 1 で仮数部が 0 でない場合です。「0.0/0.0」、「∞ / ∞」、「∞ - ∞」等、結果が数値に対応しない演算の結果として得られます。

浮動小数点型の表現する値を決定する条件を表 4.4 に示します。

表 4.4 浮動小数点型の表現する値の種類

仮数部	指数部		
	0	0 でも全ビット 1 でもない	全ビット 1
0	0	正規化数	無限大
0 以外	非正規化数		非数

注 非正規化数は、正規化数で表現できない範囲の絶対値の小さな浮動小数点型を表現しますが、正規化数と比較して有効桁数が少なくなっています。したがって、演算の結果あるいは途中結果が非正規化数となる場合、結果の有効桁数は保証しません。
denormalize=off を指定した場合、非正規化数は 0 として扱います。
denormalize=on を指定した場合、非正規化数は非正規化数のまま扱います。

(e) float 型

float 型の内部表現は、1 ビットの符号部、8 ビットの指数部、23 ビットの仮数部からなります。

(i) 正規化数

符号部は、0(正) または 1(負) で、値の符号を示します。
指数部は、1 ～ 254(2^8-2) の値をとります。実際の指数は、この値から 127 を引いた値で、その範囲は -126 ～ 127 です。
仮数部は、0 ～ 2^{23}-1 の値をとります。実際の仮数は、2^{23} のビットを 1 と仮定し、その直後に小数点があるものとして解釈します。
正規化数の表現する値は、

$$(-1)^{<\text{符号部}>} \times 2^{<\text{指数部}>-127} \times (1 + <\text{仮数部}> \times 2^{-23})$$

となります。

例

31	30 ～ 23	22 ～ 0
1	10000000	11000000000000000000000

符号：-
指数：$10000000_{(2)}$ - 127 = 1 $_{(2)}$ は 2 進数を意味します。
仮数：$1.11_{(2)}$ = 1.75
値： $-1.75 \times 2^1 = -3.5$

(ii) 非正規化数

符号部は 0(正) または 1(負) で、値の符号を示します。
指数部は 0 で、実際の指数は -126 になります。
仮数部は、1 ～ 2^{23}-1 で、実際の仮数は、2^{23} のビットを 0 と仮定し、その直後に小数点があるものとして解釈します。
非正規化数の表現する値は、

$$(-1)^{<\text{符号部}>} \times 2^{-126} \times (<\text{仮数部}> \times 2^{-23})$$

となります。

例

31	30 ～ 23	22 ～ 0
0	00000000	11000000000000000000000

符号：+
指数：-126
仮数：$0.11_{(2)}$ = 0.75 $_{(2)}$ は 2 進数を意味します。
値： 0.75×2^{-126}

(iii) ゼロ

符号部は 0(正) または 1(負) で、それぞれ +0.0、-0.0 を示します。
指数部、仮数部はともに 0 です。
+0.0、-0.0 は、ともに値としては 0.0 を示します。

(iv) 無限大

符号部は 0(正) または 1(負) で、それぞれ + ∞、- ∞を示します。
指数部は 255(2^8-1) です。
仮数部は 0 です。

(v) 非数

指数部は 255(2^8-1) です。
仮数部は 0 以外の値です。

注 仮数部の最上位ビットが 1 の非数を qNaN、仮数部の最上位ビットが 0 の非数を sNaN と呼びます。その他の仮数フィールドの値、および符号部については規定していません。

(f) double 型と long double 型

double 型と long double 型の内部表現は、1 ビットの符号部、11 ビットの指数部、52 ビットの仮数部からなります。

(i) 正規化数

符号部は 0(正) または 1(負) で、値の符号を示します。
指数部は、1 ～ 2046(2^{11}-2) の値をとります。実際の指数は、この値から 1023 を引いた値で、その範囲は -1022 ～ 1023 です。

仮数部は、0 ～ 2^{52}-1 の値となります。実際の仮数は、2^{52} のビットを 1 と仮定し、その直後に小数点があるものとして解釈します。
正規化数の表現する値は、
$(-1)^{<符号部>} \times 2^{<指数部>-1023} \times (1 + <仮数部> \times 2^{-52})$
となります。

例

63	62 ～ 52	51 ～ 0
0	01111111111	11100000000000000000000000000000000000000000000000000000

符号：+
指数：$1111111111_{(2)}$ - 1023 = 0　(2) は 2 進数を意味します。
仮数：$1.111_{(2)}$ = 1.875
値：　1.875×2^0 = 1.875

(ii) 非正規化数
符号部は 0(正) または 1(負) で、値の符号を示します。
指数部は 0 で、実際の指数は -1022 になります。
仮数部は、1 ～ 2^{52}-1 で、実際の仮数は、2^{52} のビットを 0 と仮定し、その直後に小数点があるものとして解釈します。
非正規化数が表現する値は、
$(-1)^{<符号部>} \times 2^{-1022} \times (<仮数部> \times 2^{-52})$
となります。

例

63	62 ～ 52	51 ～ 0
1	00000000000	11100000000000000000000000000000000000000000000000000000

符号：-
指数：-1022
仮数：$0.111_{(2)}$ = 0.875　(2) は 2 進数を意味します。
値：　0.875×2^{-1022}

(iii) ゼロ
符号部は 0(正) または 1(負) で、それぞれ +0.0、-0.0 を示します。
指数部、仮数部はともに 0 です。
+0.0、-0.0 は、ともに値としては 0.0 を示します。

(iv) 無限大
符号部は 0(正) または 1(負) で、それぞれ + ∞、- ∞を示します。
指数部は 2047(2^{11}-1) です。
仮数部は 0 です。

(v) 非数
指数部は 2047(2^{11}-1) です。
仮数部は 0 以外の値です。

例　仮数部の最上位ビットが 1 の非数を qNaN、仮数部の最上位ビットが 0 の非数を sNaN と呼びます。その他の仮数フィールドの値、および符号部については規定していません。

4.1.6 演算子の評価順序

式の中に複数の演算子がある場合、それらの演算子の評価順序は、優先順位と「右」または「左」で表わされる結合性によって決まります。
各演算子の優先順位と結合性を表 4.5 に示します。

表 4.5　演算子の優先順位と結合性

優先順位	演算子	結合性	適用される式
1	++ -- (後置) ( ) [ ] -> .	左	後置式
2	++ -- (前置) ! ~ + - * & sizeof	右	単項式
3	(型名)	右	キャスト式
4	* / %	左	乗除式

優先順位	演算子	結合性	適用される式
5	+ -	左	加減式
6	<< >>	左	ビット単位のシフト式
7	< <= > >=	左	関係式
8	== ! =	左	等価式
9	&	左	ビット単位の AND 式
10	^	左	ビット単位の排他 OR 式
11	|	左	ビット単位の OR 式
12	&&	左	論理 AND 式
13	| |	左	論理 OR 式
14	? :	右	条件式
15	= += -= *= /= %= <<= >>= &= |= ^=	右	代入式
16	,	左	カンマ式

4.1.7 準拠する言語仕様

(1) C 言語仕様 (lang=c オプション選択時 **)**
ANSI/ISO 9899-1990 American National Standard for Programming Languages -C

(2) C 言語仕様 (lang=c99 オプション選択時 **)**
ISO/IEC 9899:1999 INTERNATIONAL STANDARD Programming Languages - C

(3) C++ 仕様 (lang=cpp オプション選択時 **)**
Microsoft(R) Visual C/C++ 6.0 と互換性のある言語仕様に基づいています。

4.2 拡張言語仕様

この節では、CC-RX で拡張されている言語仕様について説明します。
拡張仕様には、#pragma、キーワード、組み込み関数、およびセクションアドレス演算子があります。

4.2.1 マクロ名

次にサポートしているマクロ名を示します。

表 4.6 サポートしているマクロ

	マクロ名	値	オプション
1	__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）。	―
2	__FILE__	仮定されたソース・ファイルの名前（文字列定数）。	―
3	__LINE__	その時点でのソース行の行番号（10 進数）。	―
4	__STDC__	1	―
5	__STDC_HOSTED__	1	lang=c99
6	__STDC_VERSION__	199409L(lang=c*1 時) 199901L(lang=c99 時)	lang=c*1 lang=c99
7	__STDC_IEC_559__	1	lang=c99
8	__STDC_IEC_559_COMPLEX__	1	lang=c99
9	__STDC_ISO_10646__	199712L	lang=c99
10	__cplusplus	1	lang=cpp*2 lang=ecpp
11	__TIME__	ソース・ファイルの翻訳時間（“hh:mm:ss” の形式をもつ文字列定数。	―
12	#define __RXV1 #define __RXV2 #define __RXV3 【V3.00.00 以降】	1 1 1	isa=rxv1 *3 isa=rxv2 *3 isa=rxv3 *3
13	#define __BIG #define __LIT	1 1	endian=big endian=little
14	#define __DBL4 #define __DBL8	1 1	dbl_size=4 dbl_size=8
15	#define __INT_SHORT	1	int_to_short
16	#define __SCHAR #define __UCHAR	1 1	signed_char unsigned_char
17	#define __SBIT #define __UBIT	1 1	signed_bitfield unsigned_bitfield

	マクロ名	値	オプション
18	#define __ROZ #define __RON	1 1	round=zero round=nearest
19	#define __DOFF #define __DON	1 1	denormalize=off denormalize=on
20	#define __BITLEFT #define __BITRIGHT	1 1	bit_order=left bit_order=right
21	#define __AUTO_ENUM	1	auto_enum
22	#define __FUNCTION_LIB #define __INTRINSIC_LIB	1 1	library=function library=intrinsic
23	#define __FPU	1	fpu
24	#define __RENESAS__ *4	1	—
25	#define __RENESAS_VERSION__ *4	0x*XXYYZZ*00 *5	—
26	#define __RX *4	1	—
27	#define __PIC	1	pic
28	#define __PID	1	pid
29	#define __RX600 #define __RX200	1 1	cpu=rx600 cpu=rx200
30	#define __CCRX__ *4【V2.03.00 以降】	1	—
31	#define __RX_ISA_VERSION__ 【V3.00.00 以降】	1 2 3	isa=rxv1 *3 isa=rxv2 *3 isa=rxv3 *3
32	#define __DPFPU 【V3.01.00 以降】	1	dpfpu
33	#define __TFU 【V3.01.00 以降】	1 2 （tfu_version=v2 の場合）	tfu=intrinsic tfu=intrinsic,mathlib
34	#define __TFU_MATHLIB 【V3.01.00 以降】	1 2 （tfu_version=v2 の場合）	tfu=intrinsic,mathlib

注 1. 拡張子が .c であるファイルを、lang オプションの指定なくコンパイルした場合も含みます。

注 2. 拡張子が .cpp, .cp または cc であるファイルを、lang オプションの指定なくコンパイルした場合も含みます。

注 3. 環境変数 ISA_RX による指定を含みます。

注 4. オプションに関わらず常に定義されます。

注 5. コンパイラのバージョンが V*XX.YY.ZZ* の場合、__RENESAS_VERSION__ の値は 0x*XXYYZZ*00 となります。
例) V3.01.00 の場合、#define __RENESAS_VERSION__ 0x03010000

4.2.2 キーワード

CC-RX では、拡張機能を実現するために次の字句をキーワード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-RX で追加されているキーワード一覧を示します。

表 4.7 キーワード一覧

	キーワード名	機能
1	#pragma STDC CX_LIMITED_RANGE #pragma STDC FENV_ACCESS #pragma STDC FP_CONTRACT	予約キーワード。C99 言語選択時のみ有効。 (C99 言語の文法確認のみ行い、内容は無視します。)
2	#pragma キーワード	言語拡張機能を提供します。 「4.2.3 #pragma 指令 」を参照ください。
3	__evenaccess	変数の型のサイズのアクセスを保証
4	far _far near _near	予約キーワード (型名として認識しますが、無視します。)
5	_RAM_BASE	予約キーワード ただし、-base=ram オプション指定時のみ。
6	_ROM_BASE	予約キーワード ただし、-base=rom オプション指定時のみ。
7	_PID_TOP	予約キーワード ただし、-pid オプション指定時のみ。
8	_builtin_xxx	予約キーワード _builtin_ から始まる名前を意味します。

4.2.3 #pragma 指令

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。

(1) セクション切り替え指定
コンパイラの出力するセクション名を切り替えます。
なお、記述方法の詳細については、「(1) セクション切り替え記述」を参照してください。

```
#pragma section [< セクション種別 >] [ △ < 変更セクション名 >]
< セクション種別 >: { P | C | D | B }
```

(2) スタックセクション作成
スタックセクションを作成します。
なお、記述方法の詳細については、「(2) スタックセクション作成記述」を参照してください。

```
#pragma stacksize {si=< 定数 > | su=< 定数 >}
```

(3) 割り込み関数作成
割り込み関数を作成します。
なお、記述方法の詳細については、「(3) 割り込み関数作成記述」を参照してください。

```
#pragma interrupt [(]< 関数名 >[( <割り込み仕様> [,…])][,…][)]
```

(4) 関数インライン展開
関数をインライン展開します。
なお、記述方法の詳細については、「(4) 関数のインライン展開記述」を参照してください。

```
#pragma inline [(]< 関数名 >[,…][)]
```

(5) 関数インライン展開抑止
関数をインライン展開抑止します。
なお、記述方法の詳細については、「(4) 関数のインライン展開記述」を参照してください。

```
#pragma noinline [(]< 関数名 >[,…][)]
```

(6) アセンブリ記述関数インライン展開
アセンブラ埋め込みインライン関数を作成します。
なお、記述方法の詳細については、「(5) アセンブラ記述関数インライン展開」を参照してください。

```
#pragma inline_asm[(]< 関数名 >[,…][)]
```

(7) エントリ関数指定
エントリ関数を指定します。
なお、記述方法の詳細については、「(6) エントリ関数指定」を参照してください。

```
#pragma entry[(]< 関数名 >[)]
```

(8) ビットフィールド並び順指定
ビットフィールド並び順を指定します。
なお、記述方法の詳細については、「(7) ビットフィールド並び順指定」を参照してください。

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

(9) 構造体 / クラスメンバの 1 バイトアライメント指定
構造体 / クラスメンバのアライメントを 1 バイトにします。
なお、記述方法の詳細については、「(8) 構造体 / クラスメンバのアライメント指定」を参照してください。

```
#pragma pack
```

(10) 構造体 / クラスメンバのデフォルトアライメント指定
構造体 / クラスメンバのアライメントをメンバのアライメント数にします。
なお、記述方法の詳細については、「(8) 構造体 / クラスメンバのアライメント指定」を参照してください。

```
#pragma unpack
```

(11) 構造体 / クラスメンバのオプションアライメント指定
構造体 / クラスメンバのアライメントをオプション指定にします。
なお、記述方法の詳細については、「(8) 構造体 / クラスメンバのアライメント指定」を参照してください。

```
#pragma packoption
```

(12) 変数の絶対アドレス割り付け指定
変数を絶対アドレスに割り付けます。
なお、記述方法の詳細については、「(9) 変数の絶対アドレス割り付け指定」を参照してください。

```
#pragma address [(]< 変数名 >=< 絶対アドレス >[,…][)]
```

(13) 初期値のエンディアン指定
初期値のエンディアン指定をします。
なお、記述方法の詳細については、「(10) 初期値のエンディアン指定」を参照してください。

```
#pragma endian [{big | little}]
```

(14) 分岐先 4 バイト整合指定
分岐先を 4 バイト整合する関数を指定します。
なお、記述方法の詳細については、「(11) 分岐先の命令実行向け整合を行う関数の指定」を参照してください。

```
#pragma instalign4 [(]< 関数名 >[( <分岐先の種類> )][,…][)]
```

(15) 分岐先 8 バイト整合指定

分岐先を 8 バイト整合する関数を指定します。
なお、記述方法の詳細については、「(11) 分岐先の命令実行向け整合を行う関数の指定」を参照してください。

```
#pragma instalign8 [(]< 関数名 >[( <分岐先の種類> )][,…][)]
```

(16) 分岐先整合なし指定
分岐先を整合しない関数を指定します。
なお、記述方法の詳細については、「(11) 分岐先の命令実行向け整合を行う関数の指定」を参照してください。

```
#pragma noinstalign [(]< 関数名 >[,…][)]
```

(17) スタック破壊検出コードを生成する関数の指定【Professional 版のみ】【V2.04.00 以降】
スタック破壊検出コードを生成します。
なお、記述方法の詳細については、「(12) スタック破壊検出コードを生成する関数の指定【Professional 版のみ】【V2.04.00 以降】」を参照してください。

```
#pragma stack_protector [(] 関数名 [(num=< 整数値 >)] [)]
```

(18) スタック破壊検出コード生成なし指定【Professional 版のみ】【V2.04.00 以降】
スタック破壊検出コードを生成しません。
なお、記述方法の詳細については、「(12) スタック破壊検出コードを生成する関数の指定【Professional 版のみ】【V2.04.00 以降】」を参照してください。

```
#pragma no_stack_protector [(] 関数名 [)]
```

4.2.4 拡張仕様の使用方法

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。

- セクション切り替え記述
- スタックセクション作成記述
- 割り込み関数作成記述
- 関数のインライン展開記述
- アセンブラ記述関数インライン展開
- エントリ関数指定
- ビットフィールド並び順指定
- 構造体 / クラスメンバのアライメント指定
- 変数の絶対アドレス割り付け指定
- 初期値のエンディアン指定
- 分岐先の命令実行向け整合を行う関数の指定
- スタック破壊検出コードを生成する関数の指定

(1) セクション切り替え記述

```
#pragma section [< セクション種別 >] [ △ < 変更セクション名 >]
< セクション種別 >: { P | C | D | B }
```

コンパイラの出力するセクション名を切り替えます。
セクション種別と変更セクションを指定した場合、セクション種別が P であればその #pragma 宣言以降に記述された関数のセクション名を変更します。セクション種別が C、D、または B の場合は、その #pragma 宣言以降に実体を定義した全てのセクション名を変更します。
変更セクション名のみを指定した場合、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、各デフォルトセクション名の後に < 変更セクション名 > の文字列を追加したセクション名となります。

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。
各セクション種別のデフォルトセクション名は、section オプションの指定があればそれに従います。ない場合はセクション種別名をそのまま用います。
【V3.02.00 未満】#pragma section は関数定義の外に記述しなければなりません。
【V3.02.00 以降】#pragma section を関数定義の中にも記述することができます。関数の開き括弧及び閉じ括弧は本 #pragma の有効範囲に影響しません。
【V3.02.00 以降】関数内 static 変数、静的データメンバ及びそれらの初期値が所属するセクションの名前にも作用します。ただし、次のセクションの名前には作用しません。

- 特殊化されていない関数テンプレートの中に記述した関数内 static 変数及びその初期値が所属するセクション
- クラステンプレートの中に記述した静的データメンバ及びその初期値が所属するセクション

例 1. セクション名とセクション種別を指定した場合

```
#pragma section B Ba
int i; // Ba セクションに配置
void func(void)
{
( 省略 )
}

#pragma section B Bb
int j; // Bb セクションに配置
void sub(void)
{
( 省略 )
}
```

例 2. セクション種別を省略した場合

```
#pragma section abc
int a; // Babc セクションに配置
const int c=1; // Cabc セクションに配置

void f(void)// Pabc セクションに配置

{
    a=c;
}


#pragma section
int b;// B セクションに配置

void g(void)// P セクションに配置
{
    b=c;
}
```

例 3. 静的クラスメンバへの指定

```
/*
** クラスメンバ宣言
*/

class A {
  private:
    // 初期値なし
#pragma section DATA
    static int data_;
#pragma section
    // 初期値あり
#pragma section TABLE
    static int table_[2];
#pragma section
};

/*
** 実体定義
*/

// 初期値なし
#pragma  section  DATA
int A::data_;
#pragma section

// 初期値あり
#pragma section TABLE
int A::table_[2] = { 0,  1 };
#pragma section
```

例 4. 関数内 static 変数のセクション指定【V3.02.00 以降】

```
void test1(void) {
#pragma section B B1
  static int b1;  // B1
#pragma section B B2
  static int b2;  // B2
}

// 有効範囲は関数定義の括弧に影響を受けない。
int b3;  // B2

void test2(void) {
  static int b4;  // B2
#pragma section
  static int b5;  // B
}
```

次の項目のセクション名は変更できません。section オプションを使用してください。
(1) 文字列リテラルおよび集成体の動的初期化で用いる初期化子
(2) switch 文の分岐テーブル
1 ファイルあたりの #pragma section で指定できるセクション数は最大 2045 個です。
静的クラスメンバ変数のセクションを指定する場合は、クラスのメンバ宣言と実体の定義の両方または実態の定義時のみに #pragma section の指定が必要になります。

例

```
/*
 ** クラスメンバ宣言
 */
class A
{
private:
  // 初期値なし
    #pragma section DATA
    static int data_;
    #pragma section

  // 初期値あり
    #pragma section TABLE
    static int table_[2];
    #pragma section
};

/*
** 実体定義
*/

// 初期値なし
#pragma section DATA
int A::data_;
#pragma section

// 初期値あり
#pragma section TABLE
int A::table_[2]={0, 1};
#pragma section
```

(2) スタックセクション作成記述

```
#pragma stacksize {si=< 定数 > | su=< 定数 >}
```

si=< 定数 > を指定した場合、セクション名 SI、サイズ < 定数 > のスタックとして使用するデータセクションを作成します。
su=< 定数 > を指定した場合、セクション名 SU、サイズ < 定数 > のスタックとして使用するデータセクションを作成します。

例 C ソース：

```
#pragma stacksize si=100
#pragma stacksize su=200
```

コード展開例：

```
.SECTION    SI,DATA,ALIGN=4
.BLKB       100
.SECTION    SU,DATA,ALIGN=4
.BLKB       200
```

si, su 指定はファイル内でそれぞれ 1 回しか指定できません。
< 定数 > は必ず 4 の倍数を指定してください。
< 定数 > に記述できる値の範囲は、4 から 2147483644(0x7ffffffc) までです。

(3) 割り込み関数作成記述

```
#pragma interrupt [(]< 関数名 >[( <割り込み仕様> [,…])][,…][)]
```

#pragma interrupt を用いて割り込み関数となる関数を宣言します。
関数名には、グローバル関数および静的関数メンバを指定できます。
割り込み仕様の一覧を表 4.8 に示します。

表 4.8 割り込み仕様の一覧

	項目	形式	オプション	指定内容
1	ベクタテーブル指定	vect=	< ベクタ番号 >	割り込み関数のアドレスを配置するベクタ番号
2	高速割り込み指定	fint	なし	高速割り込みに使用する関数の指定 RTFI 命令でリターン
3	割り込み関数レジスタ制限指定	save	なし	割り込み関数内で使用するレジスタの本数を制限し、退避・回復の数を減らす
4	多重割り込み許可指定	enable	なし	関数の先頭で PSW の I フラグを 1 にし、多重割り込みを許可する
5	アキュムレータ保存指定	acc	なし	割り込み関数内でアキュムレータを退避・回復する
6	アキュムレータ非保存指定	no_acc	なし	割り込み関数内でアキュムレータを退避・回復しない
7	レジスター括退避機能の利用【V3.01.00 以降】	bank=	< バンク番号 >	レジスター括退避機能を利用する 退避先のバンク番号を指定する
8	三角関数演算器の出力保存指定【V3.05.00 以降】	tfu	なし	割り込み関数内で三角関数演算器の出力を退避・回復する
9	三角関数演算器の出力非保存指定【V3.05.00 以降】	no_tfu	なし	割り込み関数内で三角関数演算器の出力を退避・回復しない

#pragma interrupt を用いて宣言した関数は、関数の処理の前後で全レジスタを保証 (関数入口 / 出口において関数内で使用する全レジスタを退避・回復) し、通常 RTE 命令でリターンします。
割り込み仕様を指定しない場合は単純な割り込み関数として処理します。

ベクタテーブルを指定した場合 (vect=) は C$VECT セクション内の指定したベクタテーブル番号位置にその関数アドレスを設定します。
【V3.00.00 以降】最適化リンケージエディタに -split_vect オプションを指定した場合は、C$VECT セクションはベクタテーブル番号ごとに分割され、個々のセクションは C$VECT< ベクタテーブル番号 > という名前を持ちます。

高速割り込み指定 (fint) をした場合は、RTFI 命令でリターンします。また、fint_register オプションを指定した場合は、オプションで指定したレジスタを退避、回復せずに割り込み関数で使用します。

割り込み関数レジスタ制限指定 (save) をした場合は、割り込み関数内で使用するレジスタの本数を R1 ～ R5、および R14 ～ R15 に制限します。R6 ～ R13 は割り込みで使用しないため、退避・回復命令は生成しません。

多重割り込み許可指定 (enable) をした場合は、割り込み関数の先頭で PSW の I フラグを 1 にし、多重割り込みを許可します。

アキュムレータ保存 (acc) を指定した場合で、#pragma interrupt を指定された関数から別の関数を呼び出している、または関数内でアキュムレータを書き換える命令を使っている場合は、アキュムレータを退避・回復する命

令を生成します。なお、ISA(*1) に RXv1 を選択した場合は、ACC を退避・回復します。ISA(*1) に RXv1 以外を選択した場合は、ACC0 と ACC1 を退避・回復します。

アキュムレータ非保存 (no_acc) を指定した場合は、ACC, ACC0, ACC1 のいずれに対しても退避・回復する命令を生成しません。
acc および no_acc のどちらの指定もない場合は、コンパイルオプション -save_acc の指定に従います。

【V3.01.00 以降】bank= ＜バンク番号＞を指定した場合は、必要に応じ SAVE, RSTR 命令を用いたレジスター括退避機能を利用します。一括退避を実施する際の退避先は＜バンク番号＞で指定した番号のレジスタ退避バンクになります。各製品のユーザーズマニュアルハードウェア編を参照し、実在するバンク番号を指定してください。
なお、-save_acc オプション未指定時、あるいはアキュムレータ非保存（no_acc）を同時に指定した場合も、レジスター括退避機能が利用される際は ACC0, ACC1 も含め退避・回復されます。ISA(*1) に RXv1 または RXv2 を選択時に、bank= ＜バンク番号＞を指定するとエラーとなります。また bank= ＜バンク番号＞を指定する場合はアセンブラオプション -bank を指定してください。ルネサス統合開発環境を使用する場合は、レジスタ退避バンクを持つマイコンの選択により、自動で -bank が指定されます。

【V3.05.00 以降】三角関数演算器の出力保存 (tfu) を指定した場合で、#pragma interrupt を指定された関数から別の関数を呼び出している、または関数内で三角関数演算器を使っている場合は、三角関数演算器の出力 (DTSR0, DTSR1) を退避・回復する命令を生成します。
三角関数演算器の出力非保存 (no_tfu) を指定した場合は、三角関数演算器の出力を退避・回復する命令を生成しません。
割り込み仕様 tfu および no_tfu は -tfu_version=v2 が指定されている場合にのみ指定できます。tfu および no_tfu のどちらの指定もない場合は、コンパイルオプション -nosave_tfu の指定に従います。

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

割り込み関数の返却値の型は void のみです。return 文の返却値を指定することはできません。指定があった場合はエラーを出力します。

例 1. 正しい宣言と誤った宣言の例

```
#pragma interrupt (f1, f2)
void f1(){…}// 正しい宣言です。
int f2(){…}// 返却値の型が void ではないのでエラーになります。
```

例 2. 通常の割り込み関数の例
C ソース：

```
#pragma interrupt func
void func(){ .... }
```

出力コード：

```
_func:
    PUSHM R1-R3; 関数内で使用しているレジスタを退避
    ....
    (R1,R2,R3 を関数内で使用 )
    ....
    POPM R1-R3; 入口で退避したレジスタを回復
    RTE
```

例 3. 関数呼び出しがある割り込み関数の例
関数内で使用しているレジスタに加えて、関数呼び出し前後で保証しないレジスタについても、割り込み関数の入口で退避し、出口で回復します。

C ソース：

```
#pragma interrupt func
void func(){
  ...
  sub();
  ...
}
```

出力コード：

```
_func:
    PUSHM R14-R15
    PUSHM R1-R5
    ...
    BSR _sub
    ...
    POPM R1-R5
    POPM R14-R15
    RTE
```

例 4. 割り込み仕様 fint を使用した場合の例
C ソース：fint_register=2 オプションを指定してコンパイル

```
#pragma interrupt func1(fint)
void func1(){ a=1; }    //  割り込み関数
void func2(){ a=2; }    //  通常関数
```

出力コード：

```
_func1:
    PUSHM R1-R3 ;  関数内で使用しているレジスタを退避
    ...         ; ( 但し、R12,R13 は退避しない )
    ...
    (R1,R2,R3,R12,R13 を関数内で使用 )
    ...
    POPM R1-R3  ;  入口で退避したレジスタを回復
    RTFI

_func2:
    ...         ; #pragma interrupt fint 指定した関数以外では、
    ...         ; R12,R13 を使用しないコードを生成する
    RTE
```

例 5. 割り込み仕様 acc を使用した場合の例
C ソース：

```
void func5(void);
#pragma interrupt accsaved_ih(acc)      /* "acc" を指定 */
void accsaved_ih(void)
{
     func5();
}
```

出力コード :

```
_accsaved_ih:
      PUSHM R14-R15
      PUSHM R1-R5
      MVFACMI R4
      SHLL #10H, R4
      MVFACHI R5
      PUSHM R4-R5
      BSR _func5
      POPM R4-R5
      MVTACLO R4
      MVTACHI R5
      POPM R1-R5
      POPM R14-R15
      RTE
```

[備考]

- RX 命令セットの仕様のため、ISA(*1) に rxv1 を指定した場合、acc フラグで退避・回復できるのは ACC の上位 48 ビットに限られ、ACC の下位 16 ビットは退避・回復されません。
- 各割り込み仕様は、小文字のみ指定できます。大文字を指定してもエラーになります。
- 割り込み仕様として vect を使った場合、指定の無い空きベクタのアドレスは 0 になります。このアドレスは、最適化リンケージエディタで任意のアドレスやシンボルに変更することができます。詳しくは、VECT および VECTN オプションの項目を参照してください。
- #pragma interrupt の関数には、引数 (仮引数) を定義することはできません。定義してもエラーにはなりませんが、引数から値を読み出しても、不定値が返されます。
- - save フラグと fint フラグを同時に有効にした場合は、R1 ～ R5、および R14 ～ R15 のほか、fint_register オプションで確保したレジスタもコード生成に使用します。

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

＜ acc、no_acc の用途＞

acc、no_acc は次のような用途を考慮したものです。

- アキュムレータの補償を save_acc で行う場合の割り込み応答速度低下の対策 (no_acc)
  既存の割り込み関数でアキュムレータの補償には save_acc オプションが有効ですが、割り込み応答速度が悪化することがあるため、不要な ACC の退避・回復を関数単位で抑止する手段として no_acc を用意します。
- アキュムレータの退避・回復をソースコードで制御 (acc と no_acc)
  アキュムレータの退避・回復の考慮が完了している割り込み関数には、明示的に acc、no_acc を選択しておくことで、ソースプログラムでアキュムレータの退避・回復を save_acc に依存せずに定義できます。

例 6. 割り込み仕様 bank を使用した場合の例

C ソース :

```
#pragma interrupt func(bank=3)      /* "bank=3" を指定 */
void func(void)
{
      ...
}
```

出力コード :

```
_func:
      SAVE #03H ;   レジスタ退避バンク ( バンク番号 3) への一括退避命令
      ...
      RSTR #03H ;   レジスタ退避バンク ( バンク番号 3) からの一括回復命令
      RTE
```

例 7. 割り込み仕様 vect と bank を併用した場合の例
C ソース：

```
#pragma interrupt func(vect=64, bank=4) /* "vect=64" と "bank=4" を指定 */
void func(void)
{
     ...
}
```

出力コード：

```
_func:
     .RVECTOR      64,_func ;  ベクタテーブル番号に 64 を登録
     SAVE #04H ;  レジスタ退避バンク ( バンク番号 4) への一括退避命令
     ...
     RSTR #04H ;  レジスタ退避バンク ( バンク番号 4) からの一括回復命令
     RTE
```

(4) 関数のインライン展開記述

```
#pragma inline [(]< 関数名 >[,…][)]
#pragma noinline [(]< 関数名 >[,…][)]
```

#pragma inline は、インライン展開する関数を宣言します。
noinline オプションが指定された場合でも、#pragma inline 指定された関数はインライン展開の対象となります。
#pragma noinline は、inline オプションの指定を抑止する関数を宣言します。
関数名には、グローバル関数および静的関数メンバを指定できます。
#pragma inline で指定した関数名の関数と関数指定子 inline(C++ 言語および C(C99) 言語) を指定した関数がインライン展開されると、その関数を呼び出したところに関数の本体が展開されます。

例 ソースファイル

```
#pragma inline(func)
static int func (int a, int b)
{
    return (a+b)/2;
}
int x;
main()
{
    x=func(10,20);
}
```

展開イメージ

```
int x;
main()
{
    int func_result;
    {
        int a_1=10, b_1=20;
        func_result=(a_1+b_1)/2;
    }
    x=func_result;
}
```

#pragma inline が指定された場合でも、以下のいずれかに該当する場合はインライン展開しません。

- 可変引数を持つ関数である。

- 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。

#pragma 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ことがあります。なお、インライン展開が抑止される際に、noscope オプションを指定すると、インライン展開されるようになる場合があります。
#pragma inline は、関数本体の定義の前に指定してください。
#pragma inline で指定した関数に対して外部定義を生成します。static 関数に #pragma inline を指定した場合、関数定義はインライン展開後に削除されます。
C++ 言語のコンパイルでは、inline が指定された関数には、外部定義を生成しません。
C(C99) 言語のコンパイルでは、inline 指定された関数には、その関数に extern 宣言がなければ、外部定義を生成しません。

(5) アセンブラ記述関数インライン展開

```
#pragma inline_asm[(]< 関数名 >[,…][)]
```

#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。

例 C ソース

```
#pragma inline_asm func
static int func(int a, int b){
   ADD R2,R1; アセンブリ記述
}
main(int *p){
*p = func(10,20);
}
```

生成コード

```
_main:
   PUSH.L R6
   MOV.L R1, R6
   MOV.L #20, R2
   MOV.L #10, R1
   ADD  R2,R1; アセンブリ記述
   MOV.L R1, [R6]
   MOV.L #0, R1
   RTSD #04H, R6-R6
```

#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 static ではない関数に対して外部定義を生成します。
アセンブラ埋め込みインライン関数内で関数の出入口で保証するレジスタ（表 9.1 レジスタ使用規則を参照）を使用する場合は、アセンブラ埋め込みインライン関数の先頭と最後でこれらのレジスタの退避・回復が必要です。

[備考]
- アセンブラ埋め込みインライン関数には、RX ファミリの命令およびテンポラリラベルだけを記述してください。テンポラリラベル以外のラベルを定義したり、アセンブラ制御命令を記述することはできません。
- アセンブラ埋め込みインライン関数の最後に RTS を記述しないでください。
関数名に関数メンバを指定することはできません。
- static 関数に #pragma inline_asm を指定した場合、関数定義はインライン展開後に削除されます。
- アセンブリ記述は、プリプロセッサの処理対象となります。このため、アセンブリ言語で使用される命令やレジスタと同じ名前のマクロ (例 : "MOV" や "R5" など) を #define でマクロ定義する場合はご注意ください。
- スタック情報ファイルは、#pragma inline_asm のアセンブリ記述内ではスタックが消費されないものとして扱います。アセンブリ記述内に R0 を操作するコードを含める場合はご注意ください。

(6) エントリ関数指定

```
#pragma entry[(]< 関数名 >[)]
```

< 関数名 > で指定した関数をエントリ関数として扱います。
エントリ関数では、レジスタの退避・回復コードを一切作成しません。
#pragma stacksize 宣言があると、関数先頭でスタックポインタの初期設定コードを出力します。
base オプションを指定した場合、オプションで指定したベースレジスタへの設定を行います。

例 C ソース：-base=rom=R13 を指定

```
#pragma stacksize su=100
#pragma entry INIT
void INIT() {
:
}
```

出力コード

```
.SECTION    SU,DATA,ALIGN=4
.BLKB       100
.SECTION    P,CODE
_INIT:
MVTC        (TOPOF SU + SIZEOF SU),USP
MOV.L       #__ROM_TOP,R13
```

#pragma entry 指定は、関数の宣言前に行ってください。
ロードモジュール全体でエントリ関数を複数指定することはできません。

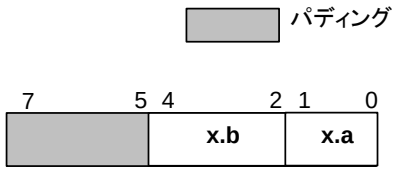
(7) ビットフィールド並び順指定

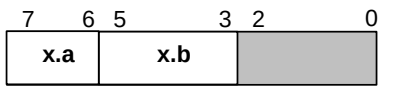
```
#pragma bit_order [{left | right}]
```

ビットフィールドの並び順の切り替えを指定します。
left を指定した場合は上位ビット側から、right を指定した場合は下位ビット側から、それぞれメンバが割り付けられます。
デフォルトの設定は right です。
left|right を省略すると、以降はオプションに従います。

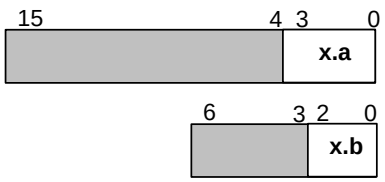
例

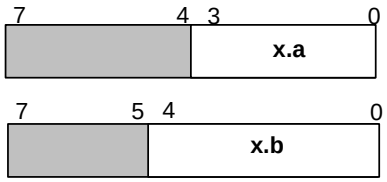
C ソース	ビット配置

#pragma bit_order right
struct tbl_r {
  unsigned char a:2;
  unsigned char b:3;
} x;
	

#pragma bit_order left
struct tbl_l {
  unsigned char a:2;
  unsigned char b:3;}
y;
	

C ソース	ビット配置

// 異なるサイズのメンバの場合
#pragma bit_order right
struct tbl_r {
  unsigned short a:4;
  unsigned char b:3;
} x;
	

// 型のサイズを超える場合
#pragma bit_order right
struct tbl_r {
  unsigned char a:4;
  unsigned char b:5;
} x;
	

(8) 構造体 / クラスメンバのアライメント指定

```
#pragma pack
#pragma unpack
#pragma packoption
```

ソースプログラム中の指定位置以降で宣言された構造体型、共用体型、およびクラス型について、メンバのアライメント数を変更します。#pragma pack も #pragma unpack も指定されていない場合または #pragma packoption 指定位置以降で宣言された構造体型、共用体型、およびクラス型については、メンバのアライメント数は pack オプションの指定に従います。#pragma とアライメント数の関係を表 4.9 に示します。

表 4.9 #pragma pack とメンバのアライメント数

メンバの型	#pragma pack	#pragma unpack	#pragma packoption または指定なし
(signed) char	1	1	1
(unsigned) short	1	2	pack オプションに従う
(unsigned) int *, (unsigned) long, (unsigned) long long, 浮動小数点型 , ポインタ型	1	4	pack オプションに従う

例

```
#pragma pack
struct S1 {
   char a;/* バイトオフセット =0*/
   int b;/* バイトオフセット =1*/
   char c;/* バイトオフセット =5*/
} ST1;/* 合計サイズ 6 バイト */

#pragma unpack
struct S2 {
   char a;/* バイトオフセット =0*/
/* 3 バイト空き領域 */
   int b;/* バイトオフセット =4*/
   char c;/* バイトオフセット =8*/
/* 3 バイト空き領域 */
} ST2;/* 合計サイズ 12 バイト */
```

構造体、共用体、クラスメンバのアライメント数は pack オプションでも指定できます。オプションと #pragma の両方が指定された場合は、#pragma の指定を優先します。

(9) 変数の絶対アドレス割り付け指定

```
#pragma address [(] < 変数名 >=< 絶対アドレス >[,...][)]
```

指定した変数を指定したアドレスに割り付けます。その際、コンパイラが指定した変数ごとにセクションを設定し、リンク時に指定した絶対アドレスに割り付けます。連続したアドレスに変数を指定した場合、それらの変数は同一セクションにします。

例 C ソース

```
#pragma address X=0x7f00
int X;
main(){
X=0;
}
```

出力コード

```
_main:
MOV.L       #0,R5
MOV.L       #7F00H,R14;
MOV.L       R5,[R14]
RTS
.SECTION    $ADDR_B_7F00,DATA
.ORG        7F00H
.glb        _X
_X:                             ; static: X
.blkl       1
```

[備考]
- #pragma address 指定は、変数の宣言前に行ってください。
- 構造体 / 共用体のメンバ、もしくは変数以外を指定した場合はエラーとなります。
- #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 が有効でも、ソースファイル内で参照のない static 変数は、最適化により削除される場合があります。
- 初期値を持ち、かつ const 修飾のない変数に対して #pragma address を適用することは推奨されません。もし該当する変数がある場合は、次の制限事項に注意してください。
  - ・この変数に対応するセクションに、最適化リンカ (rlink) の -rom オプション (ROM 領域の RAM 化) を適用することはできません。
  - ・この変数に書き込みを行ってもエラーや警告などのメッセージは表示されません。
  - ・この変数に対応するセクションは RAM に配置し、スタートアップの実行またはその前に、すべての初期値を該当する RAM 領域に書き込んでおく必要があります。
- 変数名と絶対アドレスの間の区切り文字には "=" だけでなく空白文字も使えます。

(10) 初期値のエンディアン指定

```
#pragma endian [{big | little}]
```

静的オブジェクトを格納する領域のエンディアンを指定します。
#pragma endian を記述した行から、ファイルの末尾か、または次の #pragma endian を記述した行の手前までに定義したものが対象となります。
big を指定した場合は big endian になります。オプション指定が endian=little である場合は、セクション名の後に _B をつけたセクションに配置されます。
little を指定した場合は little endian になります。オプション指定が endian=big である場合は、セクション名の後に _L をつけたセクションに配置されます。
big | little を省略すると、以降はオプションに従います。

例 endian=little オプション指定時 (デフォルト)
C ソース :

```
#pragma endian big
int A=100;/* D_B セクション */
#pragma endian
int B=200;/* D セクション */
```

出力コード :

```
    .glb   _A
    .glb   _B
    .SECTION   D,ROMDATA,ALIGN=4
_B:
    .lword   200
    .SECTION  D_B,ROMDATA,ALIGN=4
    .ENDIAN  BIG
_A:
    .lword   100
```

endian オプションと異なる #pragma endian 対象のオブジェクトに、long long 型、double 型 (dbl_size=8 オプション指定時) および long double 型 (同) の領域を含む場合は、これらの領域に対するアドレスやポインタを用いた間接的なアクセスはしないでください。この場合の動作は保証しません。
次の項目のエンディアンは変更できません。endian オプションを使用してください。
(1) 文字列リテラルおよび集成体の動的初期化で用いる初期化子
(2) switch 文の分岐テーブル
(3) 外部参照宣言されたオブジェクト（初期化式なく extern 宣言されたオブジェクト）
(4) #pragma address により指定されたオブジェクト

(11) 分岐先の命令実行向け整合を行う関数の指定

```
#pragma instalign4 [(]< 関数名 >[( ＜分岐先の種類＞ )][,…][)]
#pragma instalign8 [(]< 関数名 >[( ＜分岐先の種類＞ )][,…][)]
#pragma noinstalign [(]< 関数名 >[,…][)]
```

分岐先の命令実行向け整合を行う関数を指定します。
指定された関数に対し、#pragma instalign4 の場合は 4 バイト、#pragma instalign8 の場合は 8 バイトでそれぞれ配置アドレスを命令実行向け整合します。
#pragma noinstalign が指定された関数は、整合は行いません。
分岐先の種類は、以下から選択します (*1)。
指定なし : 関数先頭、switch 文の case および default ラベル
inmostloop: 各最内周ループの先頭、関数先頭、switch 文の case および default ラベル
loop: 各ループの先頭、関数先頭、switch 文の case および default ラベル

注 1. ここに挙げたもの以外の分岐先は、命令実行向け整合の対象ではありません。たとえば、loop を選択した場合はループの先頭は対象ですが、ループ内にあるループを構成しない if 文などの分岐先は対象ではありません。

それぞれ、指定した関数ごとに有効になることを除き、instalign4, instalign8, noinstalign オプションと機能は同じです。これらのオプションと同時に指定した場合は、#pragma の指定が優先されます。instalign4 または instalign8 を選択した関数が属するコードセクションは、そのアライメント数は 4(instalign4 の場合) または 8(instalign8 の場合) に変わります。同じコードセクション内に、instalign4 と instalign8 が指定された関数が混在する場合、そのコードセクションのアライメント数は 8 になります。
その他の内容については、instalign4, instalign8, noinstalign オプションと同様ですので、これらのオプションの項目を参照ください。

(12) スタック破壊検出コードを生成する関数の指定 【Professional 版のみ】 【V2.04.00 以降】

```
#pragma stack_protector[(] 関数名 [(num=< 整数値 >)] [,...] [)]
#pragma no_stack_protector[(] 関数名 [,...] [)]
```

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（0xFFFFFFFF 番地に向かう方向）に 4 バイトの領域を確保し、その領域に < 数値 > で指定した値を格納します。
(2) 関数の出口で、< 数値 > を格納した 4 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。
< 数値 > には 0 から 4294967295 までの 10 進数または 16 進数の整数値を指定します。< 数値 > 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。

- 返却値の型は void 型のみであり、仮引数を持たない関数です。
- 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、-stack_protector オプション、-stack_protector_all オプション、#pragma stack_protector に関わらず、スタック破壊検出コードを生成しません。
- C++ プログラム内で __stack_chk_fail 関数を定義する場合は「extern "C"」を付加してください。
- 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 関数を定義する場合は、static を指定しないでください。

#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。
#pragma stack_protector と -stack_protector オプション、-stack_protector_all オプションが同時に使用された場合は、#pragma 指定が有効になります。
同一翻訳単位内で、同じ関数に対して #pragma stack_protector と no_stack_protector を同時に指定する場合はエラーとします。
#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
#pragma inline_asm
#pragma entry

4.2.5 キーワードの使用方法

この項では、下記のキーワードの使用方法について説明します。

- 指定サイズのアクセス記述

(1) 指定サイズのアクセス記述

```
__evenaccess < 型指定子 > < 変数名 >
< 型指定子 > __evenaccess < 変数名 >
```

宣言または定義したサイズで変数をアクセスします。
変数の型のサイズでアクセスすることを保証します。
4 バイト以下のスカラ型が対象です。

例 C ソース

```
#pragma address A=0xff0178
unsigned long __evenaccess A;
void test(void)
{
    A &= ~0x20;
}
```

出力コード (__evenaccess 非指定時)

```
_test:
 MOV.L #16712056,R1
 BCLR #5,[R1]          ; 1 バイトメモリアクセス
 RTS
```

出力コード (__evenaccess 指定時)

```
_test:
 MOV.L #16712056,R1
 MOV.L [R1],R5          ; 4 バイトメモリアクセス
 BCLR  #5,R5
 MOV.L R5,[ R1]          ; 4 バイトメモリアクセス
 RTS
```

構造体や共用体単位でアクセスする場合は、__evenaccess の指定は無効です。
構造体または共用体に指定した場合、全てのメンバに __evenaccess を指定したのと同じ効果になります。その場合、4 バイト以下の整数スカラ型メンバのアクセスサイズは保証しますが、構造体または共用体単位でのアクセスサイズは保証しません。

4.2.6 組み込み関数

CC-RX では、アセンブラ命令の一部を “ 組み込み関数 ” として C ソースに記述することができます。ただし、“ アセンブラ命令そのもの ” を記述するのではなく、RX で用意した関数の形式で記述します。これらの関数を使用した場合、出力コードは通常の関数呼び出しを行わず、対応するアセンブラを出力します。

表 4.10 組み込み関数の一覧

	項目	仕様	機能	ユーザモードの制限 *1
1	最大値・最小値	signed long max(signed long data1, signed long data2)	最大値の選択	○
		signed long __max(signed long data1, signed long data2) 【V2.05.00 以降】		
2		signed long min(signed long data1, signed long data2)	最小値の選択	○
		signed long __min(signed long data1, signed long data2) 【V2.05.00 以降】		

	項目	仕様	機能	ユーザモードの制限 *1
3	バイト並べ替え	unsigned long revl(unsigned long data)	ロングワードデータをバイトリバース	○
		unsigned long __revl(unsigned long data) 【V2.05.00 以降】		
4		unsigned long revw(unsigned long data)	ロングワードデータをワード毎にバイトリバース	○
		unsigned long __revw(unsigned long data) 【V2.05.00 以降】		
5	データ交換	void xchg(signed long *data1, signed long *data2)	データ交換	○
		void __xchg(signed long *data1, signed long *data2) 【V2.05.00 以降】		
6	積和演算	long long rmpab(long long init, unsigned long count, signed char *addr1, signed char *addr2)	積和演算 (バイト)	○
		long long __rmpab(long long init, unsigned long count, signed char *addr1, signed char *addr2) 【V2.05.00 以降】		
7		long long rmpaw(long long init, unsigned long count, short *addr1, short *addr2)	積和演算 (ワード)	○
		long long __rmpaw(long long init, unsigned long count, short *addr1, short *addr2) 【V2.05.00 以降】		
8		long long rmpal(long long init, unsigned long count, long *addr1, long *addr2)	積和演算 (ロングワード)	○
		long long __rmpal(long long init, unsigned long count, long *addr1, long *addr2) 【V2.05.00 以降】		
9	回転	unsigned long rolc(unsigned long data)	キャリーを含めて 1 ビット左回転	○
		unsigned long __rolc(unsigned long data) 【V2.05.00 以降】		
10		unsigned long rorc(unsigned long data)	キャリーを含めて 1 ビット右回転	○
		unsigned long __rorc(unsigned long data) 【V2.05.00 以降】		
11		unsigned long rotl(unsigned long data, unsigned long num)	左回転	○
		unsigned long __rotl(unsigned long data, unsigned long num) 【V2.05.00 以降】		
12		unsigned long rotr (unsigned long data, unsigned long num)	右回転	○
		unsigned long __rotr(unsigned long data, unsigned long num) 【V2.05.00 以降】		

	項目	仕様	機能	ユーザモードの制限 *1
13	特殊命令	void brk(void)	BRK 命令例外	○
		void __brk(void) 【V2.05.00 以降】		
14		void int_exception(signed long num)	INT 命令例外	○
		void __int_exception(signed long num) 【V2.05.00 以降】		
15		void wait(void)	プログラム実行停止	×
		void __wait(void) 【V2.05.00 以降】		
16		void nop(void)	NOP 命令に展開	○
		void __nop(void) 【V2.05.00 以降】		
17	プロセッサ割り込み優先レベル（IPL）	void set_ipl(signed long level)	割り込み優先レベルの設定	×
		void __set_ipl(signed long level) 【V2.05.00 以降】		
18		unsigned char get_ipl(void)	割り込み優先レベルの参照	○
		unsigned char __get_ipl(void) 【V2.05.00 以降】		
19	プロセッサステータスワード（PSW）	void set_psw(unsigned long data)	PSW の設定	△
		void __set_psw(unsigned long data) 【V2.05.00 以降】		
20		unsigned long get_psw(void)	PSW の参照	○
		unsigned long __get_psw(void) 【V2.05.00 以降】		
21	浮動小数点ステータスワード（FPSW）	void set_fpsw(unsigned long data)	FPSW の設定	○
		void __set_fpsw(unsigned long data) 【V2.05.00 以降】		
22		unsigned long get_fpsw(void)	FPSW の参照	○
		unsigned long __get_fpsw(void) 【V2.05.00 以降】		
23	ユーザスタックポインタ（USP）	void set_usp(void *data)	USP の設定	○
		void __set_usp(void *data) 【V2.05.00 以降】		
24		void *get_usp(void)	USP の参照	○
		void *__get_usp(void) 【V2.05.00 以降】		
25	割り込みスタックポインタ（ISP）	void set_isp(void *data)	ISP の設定	△
		void __set_isp(void *data) 【V2.05.00 以降】		
26		void *get_isp(void)	ISP の参照	○
		void *__get_isp(void) 【V2.05.00 以降】		
27	割り込みテーブルレジスタ（INTB）	void set_intb (void *data)	INTB の設定	△
		void __set_intb(void *data) 【V2.05.00 以降】		
28		void *get_intb(void)	INTB の参照	○
		void *__get_intb(void) 【V2.05.00 以降】		

	項目	仕様	機能	ユーザモードの制限 *1
29	バックアップ PSW（BPSW）	void set_bpsw(unsigned long data)	BPSW の設定	△
		void __set_bpsw(unsigned long data) 【V2.05.00 以降】		
30		unsigned long get_bpsw(void)	BPSW の参照	○
		unsigned long __get_bpsw(void) 【V2.05.00 以降】		
31	バックアップ PC（BPC）	void set_bpc(void *data)	BPC の設定	△
		void __set_bpc(void *data) 【V2.05.00 以降】		
32		void *get_bpc(void)	BPC の参照	○
		void *__get_bpc(void) 【V2.05.00 以降】		
33	高速割り込みベクタレジスタ（FINTV）	void set_fintv(void *data)	FINTV の設定	△
		void __set_fintv(void *data) 【V2.05.00 以降】		
34		void *get_fintv(void)	FINTV の参照	○
		void *__get_fintv(void) 【V2.05.00 以降】		
35	有効桁 64bit の乗算	signed long long emul(signed long data1, signed long data2)	有効桁 64bit の符号付き乗算	○
		signed long long __emul(signed long data1, signed long data2) 【V2.05.00 以降】		
36		unsigned long long emulu(unsigned long data1, unsigned long data2)	有効桁 64bit の符号なし乗算	○
		unsigned long long __emulu(unsigned long data1, unsigned long data2) 【V2.05.00 以降】		
37	プロセッサモード（PM）	void chg_pmusr(void)	ユーザモードへの切り換え	△
		void __chg_pmusr(void) 【V2.05.00 以降】		
38	アキュムレータ（ACC）	void set_acc(signed long long data)	ACC の設定	○
		void __set_acc(signed long long data) 【V2.05.00 以降】		
39		signed long long get_acc(void)	ACC の参照	○
		signed long long __get_acc(void) 【V2.05.00 以降】		
40	割り込み許可ビットの制御	void setpsw_i(void)	割り込み許可ビットを 1 に設定	△
		void __setpsw_i(void) 【V2.05.00 以降】		
41		void clrpsw_i(void)	割り込み許可ビットを 0 に設定	△
		void __clrpsw_i(void) 【V2.05.00 以降】		

	項目	仕様	機能	ユーザモードの制限 *1
42	積和演算	long macl(short *data1, short *data2, unsigned long count)	2byte データの積和演算	○
		long __macl(short *data1, short *data2, unsigned long count) 【V2.05.00 以降】		
43		short macw1(short *data1, short *data2, unsigned long count) short macw2(short *data1, short *data2, unsigned long count)	定小数点データ向けの積和演算	○
		short __macw1(short *data1, short *data2, unsigned long count) 【V2.05.00 以降】 short __macw2(short *data1, short *data2, unsigned long count) 【V2.05.00 以降】		
44	例外ベクタテーブルレジスタ（EXTB）	void set_extb(void *data)	EXTB の設定	△
		void __set_extb(void *data) 【V2.05.00 以降】		
45		void * get_extb(void)	EXTB の参照	○
		void *__get_extb(void) 【V2.05.00 以降】		
46	ビット操作	void __bclr(unsigned char *data, unsigned long bit) 【V2.05.00 以降】	1 ビットクリア	○
47		void __bset(unsigned char *data, unsigned long bit) 【V2.05.00 以降】	1 ビットセット	○
48		void __bnot(unsigned char *data, unsigned long bit) 【V2.05.00 以降】	1 ビット反転	○
49	倍精度浮動小数点ステータスワード（DPSW）	void __set_dpsw(unsigned long data) 【V3.01.00 以降】	DPSW の設定	○
50		unsigned long __get_dpsw(void) 【V3.01.00 以降】	DPSW の参照	○
51	倍精度浮動小数点例外処理動作制御レジスタ（DECNT）	void __set_decnt(unsigned long data) 【V3.01.00 以降】	DECNT の設定	○
52		unsigned long __get_decnt(void) 【V3.01.00 以降】	DECNT の参照	○
53	倍精度浮動小数点例外プログラムカウンタ（DEPC）	void *__get_depc(void) 【V3.01.00 以降】	DEPC の参照	○

	項目	仕様	機能	ユーザモードの制限 *1
54	三角関数演算器	void __init_tfu(void) 【V3.01.00 以降】	三角関数演算器の初期化	○
55		void __sincosf(float f, float *s, float *c) 【V3.01.00 以降】	三角関数演算器を用い、正弦と余弦を同時計算（単精度）	○
56		void __atan2hypotf(float y, float x, float *a, float *h) 【V3.01.00 以降】	三角関数演算器を用い、逆正接と$\sqrt{x^2+y^2}$を同時計算（単精度）	○
57		void __sincosfx(signed long fx, signed long *s, signed long *c) 【V3.05.00 以降】	三角関数演算器を用い、正弦と余弦を同時計算（固定小数点数）	○
58		signed long __sinfx(signed long fx) 【V3.05.00 以降】	三角関数演算器を用い、正弦を計算（固定小数点数）	○
59		signed long __cosfx(signed long fx) 【V3.05.00 以降】	三角関数演算器を用い、余弦を計算（固定小数点数）	○
60		void __atan2hypotfx(signed long y, signed long x, signed long *a, signed long *h) 【V3.05.00 以降】	三角関数演算器を用い、逆正接と$\sqrt{x^2+y^2}$を同時計算（固定小数点数）	○
61		signed long __atan2fx(signed long y, signed long x) 【V3.05.00 以降】	三角関数演算器を用い、逆正接を計算（固定小数点数）	○
62		signed long __hypotfx(signed long x, signed long y) 【V3.05.00 以降】	三角関数演算器を用い、$\sqrt{x^2+y^2}$を計算（固定小数点数）	○

注 1. RX のプロセッサモードがユーザモードの場合に制限があるものを示します。
× 印の関数は特権命令例外が発生するため、ユーザモードでは使用しないでください。
△印の関数はユーザモードで実行しても効果はありません。

```
signed long max(signed long data1, signed long data2)
signed long __max(signed long data1, signed long data2) 【V2.05.00 以降】
```

[機能]
２つの入力値のうち大きい方を選択します（MAX 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
data1 入力値 1
data2 入力値 2
[リターン値]
data1 と data2 のうち大きい方の値
[例]

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
    ret = max(in1,in2);// in1 と in2 のうち大きい方の値を ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
signed long min(signed long data1, signed long data2)
signed long __min(signed long data1, signed long data2) 【V2.05.00 以降】
```

[機能]
２つの入力値のうち小さい方を選択します（MIN 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
data1 入力値 1
data2 入力値 2
[リターン値]
data1 と data2 のうち小さい方の値
[例]

```
#include < machine.h>
extern signed long ret,in1,in2;
void main(void)
{
    ret = min(in1,in2); // in1 と in2 のうち小さい方の値を ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long revl(unsigned long data)
unsigned long __revl(unsigned long data) 【V2.05.00 以降】
```

[機能]
4 バイトデータのバイト並び順をリバースします（REVL 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
data バイト並びをリバースするデータ
[リターン値]
data のバイト並びをリバースした値
[例]

```
#include <machine.h>
extern unsigned long ret,indata=0x12345678;
void main(void)
{
    ret = revl(indata);// ret=0x78563412 となる
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long revw(unsigned long data)
unsigned long __revw(unsigned long data) 【V2.05.00 以降】
```

[機能]
4 バイトデータの上位２バイトと下位２バイトでそれぞれのバイト並びをリバースします（REVW 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
data バイト並びをリバースするデータ
[リターン値]
data の上位２バイトと下位２バイトでそれぞれのバイト並びをリバースした
[例]

```
#include <machine.h>
extern unsigned long ret,indata=0x12345678;
void main(void)
{
    ret = revw(indata);// ret=0x34127856 となる
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void xchg(signed long *data1, signed long *data2)
void __xchg(signed long *data1, signed long *data2) 【V2.05.00 以降】
```

[機能]
引数が指す領域の内容を入れ替えます（XCHG 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
*data1 入力値 1
*data2 入力値 2
[リターン値]
—
[例]

```
#include <machine.h>
extern signed long *in1,*in2;
void main(void)
{
    xchg (in1,in2);// アドレス in1、アドレス in2 のデータを交換します。
}
```

[備考]
生成される XCHG 命令は、data2 が指す位置のメモリオペランドを持ちます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
long long rmpab(long long init, unsigned long count, signed char *addr1, signed char
*addr2)
long long __rmpab(long long init, unsigned long count, signed char *addr1, signed char
*addr2) 【V2.05.00 以降】
```

[機能]
初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.B 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
init 初期値
count 積和演算の回数
*addr1 乗数 1 の先頭アドレス
*addr2 乗数 2 の先頭アドレス
[リターン値]
init + S(data1[n] * data2[n]) の下位 64 ビットの結果 (n=0, 1, …, const-1)
[例]

```
#include <machine.h>
extern signed char data1[8],data2[8];
long long sum;
void main(void)
{
    sum=rmpab(0, 8, data1, data2);// 0 を初期値とし、配列 data1,data2 の
                                  // 乗算結果を加算して sum に設定する
}
```

[備考]
RMPA 命令は 80bit の範囲で結果を計算しますが、本組み込み関数では 64bit 範囲のみ扱います。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
long long rmpaw(long long init, unsigned long count, short *addr1, short *addr2)
long long __rmpaw(long long init, unsigned long count, short *addr1, short *addr2)
【V2.05.00 以降】
```

[機能]
初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.W 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
init 初期値
count 積和演算の回数
*addr1 乗数 1 の先頭アドレス
*addr2 乗数 2 の先頭アドレス
[リターン値]
init + S(data1[n] * data2[n]) の下位 64 ビットの結果 (n=0, 1, …, const-1)
[例]

```
#include <machine.h>
extern signed short data1[8],data2[8];
long long sum;
void main(void)
{
    sum=rmpaw(0, 8, data1, data2); // 0 を初期値とし、配列 data1,data2 の
                                   // 乗算結果を加算して sum に設定する
}
```

[備考]
RMPA 命令は 80bit の範囲で結果を計算しますが、本組み込み関数では 64bit 範囲のみ扱います。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
long long rmpal(long long init, unsigned long count, long *addr1, long *addr2)
long long __rmpal(long long init, unsigned long count, long *addr1, long *addr2)
【V2.05.00 以降】
```

[機能]
初期値を init、回数を count、乗数の格納されている先頭アドレスを addr1、addr2 として積和演算を行います（RMPA.L 命令に展開します）。
[ヘッダ]
<machine.h>
[引数]
init 初期値
count 積和演算の回数
*addr1 乗数 1 の先頭アドレス
*addr2 乗数 2 の先頭アドレス
[リターン値]
init + S(data1[n] * data2[n]) の下位 64 ビットの結果 (n=0, 1, …, const-1)
[例]

```
#include <machine.h>
extern signed long data1[8],data2[8];
long long sum;
void main(void)
{
    sum=rmpal(0, 8, data1, data2); // 0 を初期値とし、配列 data1,data2 の
                                   // 乗算結果を加算して sum に設定する
}
```

[備考]
RMPA 命令は 80bit の範囲で結果を計算しますが、本組み込み関数では 64bit 範囲のみ扱います。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long rolc(unsigned long data)
unsigned long __rolc(unsigned long data) 【V2.05.00 以降】
```

[機能]
C フラグを含めて 1 ビット左回転した結果を返します（ROLC 命令に展開します）。オペランドの外へ出たビットを C フラグに反映します。
[ヘッダ]
<machine.h>
[引数]
data 左回転するデータ
[リターン値]
data を C フラグを含めて左に 1 ビット回転した結果
[例]

```
#include <machine.h>
extern unsigned long ret,indata;
void main(void)
{
    ret = rolc(indata);// indata を C フラグを含めて 1 ビット左回転し
                       // ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long rorc(unsigned long data)
unsigned long __rorc(unsigned long data) 【V2.05.00 以降】
```

[機能]
C フラグを含めて 1 ビット右回転した結果を返します（RORC 命令に展開します）。オペランドの外へ出たビットを C フラグに反映します。
[ヘッダ]
<machine.h>
[引数]
data 右回転するデータ
[リターン値]
data を C フラグを含めて右に 1 ビット回転した結果
[例]

```
#include <machine.h>
extern unsigned long ret,indata;
void main(void)
{
    ret = rorc(indata);// indata を C フラグを含めて 1 ビット右回転し
                       // ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long rotl(unsigned long data, unsigned long num)
unsigned long __rotl(unsigned long data, unsigned long num) 【V2.05.00 以降】
```

[機能]
任意ビット左回転した結果を返します（ROTL 命令に展開します）。オペランドの外へ出たビットを C フラグに反映します。
[ヘッダ]
<machine.h>
[引数]
data 左回転するデータ
num 回転するビット数
[リターン値]
data を左に num ビット回転した結果
[例]

```
#include <machine.h>
extern unsigned long ret,indata;
void main(void)
{
    ret = rotl(indata, 31); // indata を 31 ビット左回転し
                            // ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long rotr (unsigned long data, unsigned long num)
unsigned long __rotr(unsigned long data, unsigned long num) 【V2.05.00 以降】
```

[機能]
任意ビット右回転した結果を返します（ROTR 命令に展開します）。オペランドの外へ出たビットを C フラグに反映します。
[ヘッダ]
<machine.h>
[引数]
data 右回転するデータ
num 回転するビット数
[リターン値]
data を右に num ビット回転した結果
[例]

```
#include <machine.h>
extern unsigned long ret,indata;
void main(void)
{
    ret = rotr(indata, 31); // indata を 31 ビット右回転し
                            // ret に設定します。
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void brk(void)
void __brk(void) 【V2.05.00 以降】
```

[機能]
BRK 命令に展開します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
    brk();// BRK 命令
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void int_exception(signed long num)
void __int_exception(signed long num) 【V2.05.00 以降】
```

[機能]
INT num 命令に展開します。
[ヘッダ]
<machine.h>
[引数]
num INT 命令番号
[リターン値]
―
[例]

```
#include <machine.h>
void main(void)
{
    int_exception(10);// INT #10 命令
}
```

[備考]
num に設定できる数は、0 ～ 255 の整数のみです。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void wait(void)
void __wait(void) 【V2.05.00 以降】
```

[機能]
WAIT 命令に展開します。
[ヘッダ]
<machine.h>
[引数]
―
[リターン値]
―
[例]

```
#include <machine.h>
void main(void)
{
    wait();// WAIT 命令
}
```

[備考]
本関数は RX のプロセッサモードがユーザモードの場合は実行しないでください。実行すると、WAIT 命令の仕様により、RX の特権命令例外が発生します。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void nop(void)
void __nop(void) 【V2.05.00 以降】
```

[機能]
NOP 命令に展開します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
    nop();// NOP 命令
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_ipl(signed long level)
void __set_ipl(signed long level) 【V2.05.00 以降】
```

[機能]
割り込みマスクレベルを変更します。
[ヘッダ]
<machine.h>
[引数]
level 設定する割り込みマスクレベル
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
    set_ipl(7);// PSW.IPL に 7 を設定
}
```

[備考]
デフォルトでは level には 0 ～ 15 の値が、-patch=rx610 を指定した場合は 0 ～ 7 の値がそれぞれ指定できます。
level が定数のとき、範囲外の値を指定した場合はエラーとなります。
本関数は RX のプロセッサモードがユーザモードの場合は使用しないでください。実行すると、MVTIPL 命令の仕様により、RX の特権命令例外が発生します。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned char get_ipl(void)
unsigned char __get_ipl(void) 【V2.05.00 以降】
```

[機能]
割り込みマスクレベルを参照します。
[ヘッダ]
<machine.h>
[引数]
―
[リターン値]
割り込みマスクレベル
[例]

```
#include <machine.h>
extern unsigned char level;
void main(void)
{
    level=get_ipl();// PSW.IPL の値を取得し level に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_psw(unsigned long data)
void __set_psw(unsigned long data) 【V2.05.00 以降】
```

[機能]
PSW を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
―
[例]

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
    set_psw(data);// PSW に data の値を設定
}
```

[備考]
RX の命令セット仕様のため、PSW の PM ビットの書き込みは無視されます。また、RX のプロセッサモードがユーザモードの場合は、PSW の IPL[3:0]、PM、U、I ビットへの書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long get_psw(void)
unsigned long __get_psw(void) 【V2.05.00 以降】
```

[機能]
PSW を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
PSW の値
[例]

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
    ret=get_psw();// PSW の値を取得し、ret に設定
}
```

[備考]
最適化の作用により、get_psw の呼び出し箇所とは違うタイミングで PSW レジスタの値が取得される場合があります。このため、何らかの演算後に、本関数の戻り値に含まれる C,Z,S および O フラグのいずれかを利用するコードを記述した場合、その動作は保証しません。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_fpsw(unsigned long data)
void __set_fpsw(unsigned long data) 【V2.05.00 以降】
```

[機能]
FPSW を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
    set_fpsw(data);// FPSW に data の値を設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long get_fpsw(void)
unsigned long __get_fpsw(void) 【V2.05.00 以降】
```

[機能]
FPSW を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
FPSW の値
[例]

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
    ret=get_fpsw();// FPSW の値を取得し、ret に設定
}
```

[備考]
最適化の作用により、get_fpsw の呼び出し箇所とは違うタイミングで FPSW レジスタの値が取得される場合があります。このため、何らかの演算後に、本関数の戻り値に含まれる CV,CO,CZ,CU,CX,CE,FV,FO,FZ,FU,FX および FS フラグのいずれかを利用するコードを記述した場合、その動作は保証しません。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_usp(void *data)
void __set_usp(void *data) 【V2.05.00 以降】
```

[機能]
USP を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_usp(data);// USP に data の値を設定
}
```

[備考]
data には 4 バイト境界のアドレスを指定してください。
1 バイト境界、もしくは 2 バイト境界のアドレスを指定した場合、プログラムの動作は保証できません。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_usp(void)
void *__get_usp(void) 【V2.05.00 以降】
```

[機能]
USP を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
USP の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_usp();// USP の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_isp(void *data)
void __set_isp(void *data) 【V2.05.00 以降】
```

[機能]
ISP を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_isp(data);// ISP に data の値を設定
}
```

[備考]
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、ISP への書き込みは無視されます。
data には 4 バイト境界のアドレスを指定してください。
1 バイト境界、もしくは 2 バイト境界のアドレスを指定した場合、プログラムの動作は保証できません。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_isp(void)
void *__get_isp(void) 【V2.05.00 以降】
```

[機能]
ISP を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ISP の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_isp();// ISP の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_intb(void *data)
void __set_intb(void *data) 【V2.05.00 以降】
```

[機能]
INTB を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_intb (data);// INTB に data の値を設定
}
```

[備考]
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、INTB への書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_intb(void)
void *__get_intb(void) 【V2.05.00 以降】
```

[機能]
INTB を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
INTB の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_intb();// INTB の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_bpsw(unsigned long data)
void __set_bpsw(unsigned long data) 【V2.05.00 以降】
```

[機能]
BPSW を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern unsigned long data;
void main(void)
{
    set_bpsw (data);// BPSW に data の値を設定
}
```

[備考]
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、BPSW への書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long get_bpsw(void)
unsigned long __get_bpsw(void) 【V2.05.00 以降】
```

[機能]
BPSW を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
BPSW の値
[例]

```
#include <machine.h>
extern unsigned long ret;
void main(void)
{
    ret=get_bpsw ();// BPSW の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_bpc(void *data)
void __set_bpc(void *data) 【V2.05.00 以降】
```

[機能]
BPC を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_bpc(data);// BPC に data の値を設定
}
```

[備考]
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、BPC への書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_bpc(void)
void *__get_bpc(void) 【V2.05.00 以降】
```

[機能]
BPC を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
BPC の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_bpc();// BPC の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_fintv(void *data)
void __set_fintv(void *data) 【V2.05.00 以降】
```

[機能]
FINTV を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
ー
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_fintv(data);// FINTV に data の値を設定
}
```

[備考]
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、FINTV への書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_fintv(void)
void *__get_fintv(void) 【V2.05.00 以降】
```

[機能]
FINTV を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
FINTV の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_fintv();// FINTV の値を取得し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
signed long long emul(signed long data1, signed long data2)
signed long long __emul(signed long data1, signed long data2) 【V2.05.00 以降】
```

[機能]
有効桁 64bit の符号付き乗算を行います。
[ヘッダ]
<machine.h>
[引数]
data1 入力値 1
data2 入力値 2
[リターン値]
符号付き乗算の結果（64bit 符号付き）
[例]

```
#include <machine.h>
extern signed long long ret;
extern signed long data1, data2;
void main(void)
{
    ret=emul(data1, data2);// data1 * data2 の値を計算し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
unsigned long long emulu(unsigned long data1, unsigned long data2)
unsigned long long __emulu(unsigned long data1, unsigned long data2)  【V2.05.00 以降】
```

[機能]
有効桁 64bit の符号なし乗算を行います。
[ヘッダ]
<machine.h>
[引数]
data1　入力値 1
data2　入力値 2
[リターン値]
符号なし乗算の結果（64bit 符号なし）
[例]

```
#include <machine.h>
extern unsigned long long ret;
extern unsigned long data1, data2;
void main(void)
{
    ret=emulu(data1, data2);// data1 * data2 の値を計算し、ret に設定
}
```

[備考]
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void chg_pmusr(void)
void __chg_pmusr(void)  【V2.05.00 以降】
```

[機能]
RX のプロセッサモードをユーザに切り換えます。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void);
void Do_Main_on_UserMode(void)
{
    chg_pmusr();  // プロセッサモードをユーザモードに切り換え
    main();       // main を実行
}
```

[備考]
本関数はリセット処理関数あるいは割り込み関数のために用意されています。それ以外の関数での使用は推奨しません。
RX のプロセッサモードがユーザモードのときは、プロセッサモードは切り替わりません。
chg_pmusr 関数の実行により、スタックは割り込みスタックからユーザスタックに切り換えが起こりますので、本関数を呼び出す関数では、必ず次の条件を守ってください。守られない場合、本関数の実行前後のスタックの相違が起こるため、コードは正常に動作しません。

- 呼び出し元に return することはできません。
- auto 変数を宣言することはできません。
- 引数を宣言することはできません。

__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_acc(signed long long data)
void __set_acc(signed long long data)  【V2.05.00 以降】
```

[機能]
ACC を設定します。
[ヘッダ]
<machine.h>
[引数]
data ACC への設定値
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
signed long long data = 0x123456789ab0000LL;
    set_acc(data);// ACC に data の値を設定
}
```

[備考]
ACC0, ACC1 はサポートしていません。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
signed long long get_acc(void)
signed long long __get_acc(void)  【V2.05.00 以降】
```

[機能]
ACC を参照します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ACC の値
[例]

```
/* get_acc と set_acc による ACC の退避 / 回復を利用したプログラムの例 */
#include <machine.h>
signed long a, b, c;
void func(void)
{
    signed long long bak_acc = get_acc(); // ACC の値を取得し、bak_acc に退避
    c = a * b; // 乗算 (ACC を破壊する )
    set_acc(bak_acc); // bak_acc で退避していた値で ACC を回復
}
```

[備考]
ACC0, ACC1 はサポートしていません。
RX 命令セットの仕様のため、ACC の下位 16 ビットは取得できません。本関数は、これらのビットには 0 を返します。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void setpsw_i(void)
void __setpsw_i(void)  【V2.05.00 以降】
```

[機能]
PSW の割り込み許可ビット (I ビット) を 1 に設定します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
    setpsw_i();// 割り込み許可ビットを 1 にする
}
```

[備考]
本関数で使用する SETPSW 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、割り込み許可ビットへの書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void clrpsw_i(void)
void __clrpsw_i(void)  【V2.05.00 以降】
```

[機能]
PSW の割り込み許可ビット (I ビット) を 0 に設定します。
[ヘッダ]
<machine.h>
[引数]
ー
[リターン値]
ー
[例]

```
#include <machine.h>
void main(void)
{
    clrpsw_i();// 割り込み許可ビットを 0 にする
}
```

[備考]
本関数で使用する CLRPSW 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、割り込み許可ビットへの書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
long macl(short *data1, short *data2, unsigned long count)
long __macl(short *data1, short *data2, unsigned long count)  【V2.05.00 以降】
```

[機能]
積和演算を行います。
2byte のデータ同士の積和演算を行い、その結果を 4byte で返します。
積和演算は、DSP 機能命令 (MULLO,MACLO,MACHI) を使用して演算します。
積和演算の途中のデータは、ACC に 48bit 長データとして保持されます。
全ての積和演算が終わったら、ACC の内容を MVFACMI 命令で取り出し、組み込み関数の戻り値とします。
本組み込み関数を使用することにより、組み込み関数を使用せずに積和演算を記述する場合よりも、高速な積和演算処理が期待できます。
2byte の整数データの積和演算をする場合に利用できます。積和演算の演算結果には、飽和処理や丸め処理はされません。

[ヘッダ]
<machine.h>

[引数]
data1　乗数 1 の先頭アドレス
data2　乗数 2 の先頭アドレス
count　積和演算の乗算回数

[リターン値]
Σ(data1[n]×data2[n]) の演算結果

[例]

```
#include <machine.h>
short data1[3] = {a1, b1, c1};
short data2[3] = {a2, b2, c2};
void mac_calc()
{
result = macl(data1, data2, 3);    /* a1*a2+b1*b2+c1*c2 の結果を求めます */
}
```

[備考]
積和演算に使用される各種 DSP 機能命令の詳細な内容を確認するには、プログラミングマニュアルを参照してください。
乗算回数が 0 の場合、組み込み関数の戻り値は 0 です。
本組み込み関数を使用する場合、ACC の内容が書き換わる割り込み処理では、ACC を退避回復してください。
ACC を退避回復する機能については、コンパイルオプションの save_acc、もしくは、拡張言語仕様の #pragma interrupt を参照してください。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
short macw1(short *data1, short *data2, unsigned long count)
short macw2(short *data1, short *data2, unsigned long count)
short __macw1(short *data1, short *data2, unsigned long count) 【V2.05.00 以降】
short __macw2(short *data1, short *data2, unsigned long count) 【V2.05.00 以降】
```

[機能]
2byte のデータ同士の積和演算を行い、その結果を 2byte で返します。
積和演算は、DSP 機能命令 (MULLO,MACLO,MACHI) を使用して演算します。
積和演算の途中のデータは、ACC に 48bit 長データとして保持されます。
全ての積和演算が終わった後に、ACC の積和演算結果に対して丸め処理を行います。
macw1 関数は "RACW #1" 命令、macw2 関数は "RACW #2" 命令で丸め処理を行います。
丸め処理の処理内容は、以下の手順になります。

- ACC の内容を、macw1 関数は 1 ビット、macw2 関数は 2 ビット、左シフトします
- ACC の下位 32 ビットの最上位ビットを 0 捨 1 入します
- ACC の上位 32 ビットを、上限値 0x00007FFF、下限値を 0xFFFF8000 として飽和処理します

最後に、MVFACHI 命令で ACC から上位 32 ビットを取得し、組み込み関数の戻り値とします。

通常、固定小数点データ同士の乗算をする場合、乗算結果の小数点位置を調整する必要があります。たとえば、Q15 形式の固定小数点データ同士の乗算の場合、乗算結果を同じ Q15 形式にするためには、乗算結果を 1 ビット左シフトする必要があります。
この小数点位置を調整するための左シフトは、RACW 命令の左シフト動作によって実現できます。そのため、2byte の固定小数点データの積和演算をする場合に、本組み込み関数を利用することにより、容易に積和演算処理を実現できます。なお、macw1 と macw2 は演算結果の丸め方法が異なりますので、演算結果に求められる精度に応じて、使用する組み込み関数を選択してください。

[ヘッダ]
<machine.h>

[引数]
data1 乗数 1 の先頭アドレス
data2 乗数 2 の先頭アドレス
count 積和演算の乗算回数

[リターン値]
積和演算の演算結果を、RACW 命令で丸めた値

[例]

```
#include <machine.h>
short data1[3] = {a1, b1, c1};
short data2[3] = {a2, b2, c2};
void mac_calc()
{
result = macw1(data1, data2, 3);
/* a1*a2+b1*b2+c1*c2 の結果を、"RACW #1" 命令で丸めた値を求めます。*/
}
```

[備考]
積和演算に使用される各種 DSP 機能命令の詳細な内容を確認するには、プログラミングマニュアルを参照してください。
乗算回数が 0 の場合、組み込み関数の戻り値は 0 です。
本組み込み関数を使用する場合、ACC の内容が書き換わる割り込み関数では、ACC を退避回復してください。
ACC を退避回復する機能については、コンパイルオプションの save_acc、もしくは、拡張言語仕様の #pragma interrupt を参照してください。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void set_extb(void *data)
void __set_extb(void *data) 【V2.05.00 以降】
```

[機能]
EXTB を設定します。
[ヘッダ]
<machine.h>
[引数]
data 設定値
[リターン値]
―
[例]

```
#include <machine.h>
extern void * data;
void main(void)
{
    set_extb (data);// EXTB に data の値を設定
}
```

[備考]
本関数は、isa オプションもしくは環境変数 ISA_RX に RXv1 以外が指定されている場合に使用できます。それ以外の場合はコンパイル時にエラーとなります。
本関数で使用する MVTC 命令の仕様により、RX のプロセッサモードがユーザモードの場合は、EXTB への書き込みは無視されます。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void *get_extb(void)
void *__get_extb(void) 【V2.05.00 以降】
```

[機能]
EXTB を参照します。
[ヘッダ]
<machine.h>
[引数]
―
[リターン値]
EXTB の値
[例]

```
#include <machine.h>
extern void * ret;
void main(void)
{
    ret=get_extb();// ENTB の値を取得し、ret に設定
}
```

[備考]
本関数は、isa オプションもしくは環境変数 ISA_RX に RXv1 以外が指定されている場合に使用できます。それ以外の場合はコンパイル時にエラーとなります。
__ で始まる名称の組み込み関数を使用する場合には、ヘッダのインクルードは不要です。

```
void __bclr(unsigned char *data, unsigned long bit) 【V2.05.00 以降】
```

[機能]
指定した 1 バイト領域の指定した 1 ビットを 0 にします（BCLR 命令に展開します）。
[ヘッダ]
―
[引数]
data 操作対象となる 1 バイト領域のアドレス。
bit 操作対象のビット位置。
[リターン値]
―
[例]

```
unsigned char *data;
void main(void)
{
    __bclr(data, 0);  // アドレス data が指示する 1 バイト領域の最下位ビットを 0 にします。
}
```

[備考]
引数 bit に指定できる値は 0 ～ 7 の整数定数のみです。
引数で指定したビットをメモリ上で直接 0 に書き換える BCLR 命令に展開します。

```
void __bset(unsigned char *data, unsigned long bit) 【V2.05.00 以降】
```

[機能]
指定した 1 バイト領域の指定した 1 ビットを 1 にします（BSET 命令に展開します）。
[ヘッダ]
―
[引数]
data 操作対象となる 1 バイト領域のアドレス。
bit 操作対象のビット位置。
[リターン値]
―
[例]

```
unsigned char *data;
void main(void)
{
    __bset(data, 0);  // アドレス data が指示する 1 バイト領域の最下位ビットを 1 にします。
}
```

[備考]
引数 bit に指定できる値は 0 ～ 7 の整数定数のみです。
引数で指定したビットをメモリ上で直接 1 に書き換える BSET 命令に展開します。

```
void __bnot(unsigned char *data, unsigned long bit) 【V2.05.00 以降】
```

[機能]
指定した 1 バイト領域の指定した 1 ビットの値を反転します（BNOT 命令に展開します）。
[ヘッダ]
ー
[引数]
data 操作対象となる 1 バイト領域のアドレス。
bit 操作対象のビット位置。
[リターン値]
ー
[例]

```
unsigned char *data;
void main(void)
{
    __bnot(data, 0);  // アドレス data が指示する 1 バイト領域の最下位ビットが 1 なら 0 に、
                      // 0 なら 1 にします。
}
```

[備考]
引数 bit に指定できる値は 0 ～ 7 の整数定数のみです。
引数で指定したビットをメモリ上で直接反転する BNOT 命令に展開します。

```
void __set_dpsw(unsigned long data) 【V3.01.00 以降】
```

[機能]
DPSW を設定します。
[ヘッダ]
ー
[引数]
data 設定値
[リターン値]
ー
[例]

```
unsigned long data;
void main(void)
{
    __set_dpsw(data);// DPSW に data の値を設定
}
```

[備考]
本関数は、dpfpu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。

```
unsigned long __get_dpsw(void) 【V3.01.00 以降】
```

[機能]
DPSW を参照します。
[ヘッダ]
―
[引数]
―
[リターン値]
DPSW の値
[例]

```
unsigned long ret;
void main(void)
{
    ret=__get_dpsw();// DPSW の値を取得し、ret に設定
}
```

[備考]
本関数は、dpfpu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。

```
void __set_decnt(unsigned long data) 【V3.01.00 以降】
```

[機能]
DECNT を設定します。
[ヘッダ]
―
[引数]
data 設定値
[リターン値]
―
[例]

```
unsigned long data;
void main(void)
{
    __set_decnt(data);// DECNT に data の値を設定
}
```

[備考]
本関数は、dpfpu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。

```
unsigned long __get_decnt(void) 【V3.01.00 以降】
```

[機能]
DECNT を参照します。
[ヘッダ]
ー
[引数]
ー
[リターン値]
DECNT の値
[例]

```
unsigned long ret;
void main(void)
{
    ret=__get_decnt();// DECNT の値を取得し、ret に設定
}
```

[備考]
本関数は、dpfpu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。

```
void *__get_depc(void) 【V3.01.00 以降】
```

[機能]
DEPC を参照します。
[ヘッダ]
ー
[引数]
ー
[リターン値]
DEPC の値
[例]

```
void *ret;
void main(void)
{
    ret=__get_depc();// DEPC の値を取得し、ret に設定
}
```

[備考]
本関数は、dpfpu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。

```
void __init_tfu(void) 【V3.01.00 以降】
```

[機能]
三角関数演算器 (v1) を初期化します。
[ヘッダ]
―
[引数]
―
[リターン値]
―
[例]

```
void main(void)
{
#if __TFU == 1
    __init_tfu();
#endif
}
```

[備考]
本関数は tfu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。
TFUv1 を使用している場合、三角関数演算器を利用する前にスタートアッププログラム等で本関数を用いた演算器の初期化を行ってください。初期化を行わない場合の動作は保証しません。
TFUv2 を使用している場合、演算器を初期化する必要はありません。本関数を呼び出すとコンパイル時にエラーとなります。

```
void __sincosf(float f, float *s, float *c) 【V3.01.00 以降】
```

[機能]
三角関数演算器を用い、正弦と余弦を同時に計算します。（単精度）
[ヘッダ]
ー
[引数]
f 正弦・余弦を求めるラジアン値
s 正弦の演算結果を格納するアドレス
c 余弦の演算結果を格納するアドレス
[リターン値]
ー
[例]

```
float f, *s, *c;
void main(void)
{
#if __TFU == 1
    __init_tfu();              // 事前に三角関数演算器の初期化が必要
#endif

    __sincosf(f, s, c);
}
```

[備考]
本関数は tfu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。
TFUv1 を使用している場合、本関数を呼び出す前に組み込み関数 __init_tfu() を用いた三角関数演算器の初期化が必要です。また、本関数はリエントラントではありません。
TFUv2 を使用している場合、演算器を初期化する必要はありません（__init_tfu() を呼び出すとコンパイル時にエラーとなります）。また、本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。

- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

```
void __atan2hypotf(float y, float x, float *a, float *h) 【V3.01.00 以降】
```

[機能]
三角関数演算器を用い、x,y の逆正接と 2 乗の和の平方根（$\sqrt{x^2+y^2}$）を同時に計算します。（単精度）
[ヘッダ]
ー
[引数]
y　y 座標（正接の分子）
x　x 座標（正接の分母）
a　y/x の逆正接の演算結果を格納するアドレス
h　2 乗の和の平方根（$\sqrt{x^2+y^2}$）演算結果を格納するアドレス
[リターン値]
ー
[例]

```
float y, x, *a, *h;
void main(void)
{
#if __TFU == 1
    __init_tfu();              // 事前に三角関数演算器の初期化が必要
#endif

    __atan2hypotf(y, x, a, h);
}
```

[備考]
本関数は tfu オプションが指定されている場合に使用できます。未指定の場合はコンパイル時にエラーとなります。
TFUv1 を使用している場合、本関数を呼び出す前に組み込み関数 __init_tfu() を用いた三角関数演算器の初期化が必要です。また、本関数はリエントラントではありません。
TFUv2 を使用している場合、演算器を初期化する必要はありません（__init_tfu() を呼び出すとコンパイル時にエラーとなります）。また、本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

逆正接の演算結果はラジアン値で、その範囲は (-π,π) です。

```
void __sincosfx(signed long fx, signed long *s, signed long *c)【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、正弦と余弦を同時に計算します。（固定小数点数）
[ヘッダ]
ー
[引数]
fx 正弦・余弦を求める角度
s 正弦の演算結果を格納するアドレス
c 余弦の演算結果を格納するアドレス
[リターン値]
ー
[例]

```
signed long fx, *s, *c;
void main(void) {
    __sincosfx(fx, s, c);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

演算結果の入力形式、入力単位、及び出力形式は FXSCIOC レジスタの設定によります。詳細はハードウェアマニュアルをご覧ください。

```
signed long __sinfx(signed long fx)【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、正弦を計算します。（固定小数点数）
[ヘッダ]
ー
[引数]
fx 正弦を求める角度
[リターン値]
正弦の演算結果
[例]

```
signed long fx, s;
void main(void) {
    s = __sinfx(fx);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

演算結果の入力形式、入力単位、及び出力形式は FXSCIOC レジスタの設定によります。詳細はハードウェアマニュアルをご覧ください。

```
signed long __cosfx(signed long fx)【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、余弦を計算します。（固定小数点数）
[ヘッダ]
ー
[引数]
fx 余弦を求める角度
[リターン値]
余弦の演算結果
[例]

```
signed long fx, c;
void main(void) {
    c = __cosfx(fx);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

演算結果の入力形式、入力単位、及び出力形式は FXSCIOC レジスタの設定によります。詳細はハードウェアマニュアルをご覧ください。

```
void __atan2hypotfx(signed long y, signed long x, signed long *a, signed long *h)
【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、逆正接と 2 乗の和の平方根（$\sqrt{x^2 + y^2}$）を同時に計算します。（固定小数点数）
[ヘッダ]
ー
[引数]
y y 座標（正接の分子）
x x 座標（正接の分母）
a y/x の逆正接の演算結果を格納するアドレス
h 2 乗の和の平方根（$\sqrt{x^2 + y^2}$）の演算結果を格納するアドレス
[リターン値]
ー
[例]

```
signed long y, x, *a, *h;
void main(void) {
    __atan2hypotfx(y, x, a, h);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

逆正接の演算結果の出力形式及び出力単位は FXATIOC レジスタの設定によります。詳細はハードウェアマニュアルをご覧ください。

2 乗の和の平方根（$\sqrt{x^2 + y^2}$）の演算結果の出力形式は Q3.29 固定です。

```
signed long __atan2fx(signed long y, signed long x)【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、逆正接を計算します。(固定小数点数)
[ヘッダ]
ー
[引数]
y y 座標(正接の分子)
x x 座標(正接の分母)
[リターン値]
y/x の逆正接の演算結果
[例]

```
signed long y, x, a;
void main(void) {
    a = __atan2fx(y, x);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

演算結果の出力形式及び出力単位は FXATIOC レジスタの設定によります。詳細はハードウェアマニュアルをご覧ください。

```
signed long __hypotfx(signed long x, signed long y)【V3.05.00 以降】
```

[機能]
三角関数演算器を用い、2 乗の和の平方根($\sqrt{x^2 + y^2}$)を計算します。(固定小数点数)
[ヘッダ]
ー
[引数]
x x 座標
y y 座標
[リターン値]
2 乗の和の平方根($\sqrt{x^2 + y^2}$)の演算結果
[例]

```
signed long x, y, h;
void main(void) {
    h = __hypotfx(x, y);
}
```

[備考]
本関数は -tfu オプションが指定され、かつ TFUv2 を使用している場合に使用できます。いずれかが該当しない場合はコンパイル時にエラーとなります。
本関数はデフォルトでリエントラントですが、次のいずれかに該当する場合はリエントラントではありません。
- -nosave_tfu を指定した場合。
- 三角関数演算器を使用するいずれかの割り込み関数で #pragma interrupt に割り込み仕様 no_tfu を指定した場合。

演算結果の出力形式は Q3.29 固定です。

4.2.7 セクションアドレス演算子

セクションアドレス演算子の一覧を表 4.11 に示します。

表 4.11 セクション演算子一覧

	セクション演算子名	説明
1	__sectop("< セクション名 >")	< セクション名 > の先頭アドレスを返します。
2	__secend("< セクション名 >")	< セクション名 > の先頭アドレスに、< セクション名 > のサイズを加算した値を返します。
3	__secsize("< セクション名 >")	< セクション名 > のサイズを返します。

[型]
__sectop の型は、void * です。
__secend の型は、void * です。
__secsize の型は、unsigned long です。

[例]
例 1: __sectop, __secend

```
#include <machine.h>
#pragma section $DSEC
static const struct {
    void *rom_s; /* 初期化データセクションの ROM 上の先頭アドレス */
    void *rom_e; /* 初期化データセクションの ROM 上の最終アドレス */
    void *ram_s; /* 初期化データセクションの RAM 上の先頭アドレス */
} DTBL[]={__sectop("D"), __secend("D"), __sectop("R")};

#pragma section $BSEC
static const struct {
    void *b_s; /* 未初期化データセクションの先頭アドレス */
    void *b_e; /* 未初期化データセクションの最終アドレス */
} BTBL[]={__sectop("B"), __secend("B")};
#pragma section
#pragma stacksize si=0x100
#pragma entry INIT
void main(void);
void INIT(void)
{
    _INITSCT();
    main();
    sleep();
}
```

例 2: __secsize

```
/* size of section B */
unsigned int size_of_B = __secsize("B");
```

[備考]
PIC/PID 機能が有効なアプリケーションの場合、__sectop および __secend はリンク時のアドレスで処理します。
PIC/PID 機能の詳細は、pic および pid オプションと、「8.5 PIC/PID 機能の利用」の項目をそれぞれ参照してください。

5. アセンブラ言語仕様

この章では、RX がサポートするアセンブリ言語仕様について説明します。アセンブリソース内で利用可能なレジスタ、命令、データタイプ等については RX ソフトウェアマニュアルを参照してください。

5.1 ソースの記述方法

この節では、ソースの記述方法、式と演算子などについて説明します。

5.1.1 記述方法

ニーモニック記述行の構成を以下に記します。
[ラベル][オペレーション [△オペランド]][コメント]
(コーディング例)

```
LABEL1:         MOV.L       [R1], R2            ; ニーモニック記述行の例です。
                            オペランド                  コメント
                オペレーション
ラベル
```

(1) ラベル
ニーモニック記述行のアドレスに対して名前を定義します。

(2) オペレーション
ニーモニック、制御命令を書きます。

(3) オペランド
オペレーションの実行対象となるものを書きます。オペランドの個数と種類はオペレーションによって決まります。オペランドを必要としないオペレーションもあります。

(4) コメント
プログラムを分かりやすくするための注釈を書きます。

5.1.2 名前

名前はアセンブリ言語ファイルの中で任意に定義し使用できます。
名前は次の種類に分けられます。

表 5.1 名前の種類

名前の種類	内容
ラベル名	アドレスを値として持つ名前
シンボル名	定数を値として持つ名前 (シンボル名には、ラベル名も含みます)
セクション名	.SECTION 制御命令で定義されるセクションの名前
ロケーションシンボル名	ロケーションシンボル '$' が記述されている行のオペレーション部の先頭アドレスを示します
マクロ名	マクロの定義名

名前の記述規則

- 名前の文字数に制限はありません。
- 名前は大文字と小文字を区別して扱います。"LAB" と "Lab" は異なる名前として扱われます。
- 名前には英数字とアンダーライン (_)、ドル ($) が使用できます。
- 名前の先頭文字に数字は使用できません。
- 名前に予約語を使用することはできません。

* ただし、セクション名にのみ予約語のフラグ名 (U,I,O,S,Z,C) を使用することができます。

5.1.3 ラベルの記述方法

ラベルを記述する場合、名前の最後に必ずコロン (:) を付けます。

- 記述例

```
LABEL1:
```

定義されているセクション名と同じ名前のシンボルを定義することはできません。セクションとシンボルを同じ名前で定義した場合、後で定義したものは A2118 エラーとなります。

5.1.4 オペレーション部の記述方法

- 記述方法

```
ニーモニック [ サイズ指定子 ( 分岐距離指定子 )]
```

- 内容
命令は、以下２つの要素から構成されます。
(1) ニーモニック ・・・ 命令の動作を示します。
(2) サイズ指定子 ・・・ 処理対象のデータサイズを指定します。

(1) ニーモニック
ニーモニックは命令の動作を示します。

例

```
MOV ・・・ 転送命令
ADD ・・・ 算術演算命令 ( 加算命令 )
```

(2) サイズ指定子
サイズ指定子は、命令コードのオペランドサイズを指定するものです。

- 記述方法

```
.size
```

- 内容
オペランドの演算サイズを指定します。正確にはその命令が処理を行うために読み出すデータのサイズを指定します。size は下記表の通りです。

表 5.2 サイズ指定子

size	内容
B	バイト (8 ビット)
W	ワード (16 ビット)
L	ロングワード (32 ビット)

size は大文字小文字いずれも可。
(例) MOV.B #0, R3 ・・・バイト指定
本指定子は、「RX ファミリソフトウェアマニュアル」の命令フォーマットで (.size) が明記されているものについてのみ指定が可能でかつ必須となります。

(3) 分岐距離指定子
分岐距離指定子は、分岐命令および相対サブルーチン分岐命令で指定します。

- 記述方法

```
.length
```

- 内容
length は下記表の通りです。

表 5.3 分岐距離指定子

length	内容	
S	3 ビット PC 前方相対	(+3 ～ +10)
B	8 ビット PC 相対	(-128 ～ +127)
W	16 ビット PC 相対	(-32768 ～ +32767)
A	24 ビット PC 相対	(-8388608 ～ +8388607)
L	レジスタ相対	(-2147483648 ～ +2147183647)

length は大文字小文字いずれも可。
(例)
BRA.W label ・・・16 ビット相対指定
BRA.L R1 ・・・レジスタ相対指定
本指定子は省略可能です。省略した場合は次の条件を全て満たした場合のみ、もっともオペコードが小さくなるように (S/B/W/A の中から) アセンブラがコードを選択します。
(1) オペランドが、レジスタ以外で記述されている場合
(2) オペランドが、アセンブル時に分岐距離が確定する分岐先である場合
(例) ラベル＋アセンブル時確定値
ラベルーアセンブル時確定値
アセンブル時確定値＋ラベル
(3) オペランドのラベルが同一セクション内で定義されている場合
また、オペランドがレジスタの場合、分岐距離指定子 L が選択されます。
条件分岐命令の場合、分岐距離が規定の範囲を超えているときは条件を反転してコードを生成します。
各命令で指定可能な分岐距離指定子は、下記表の通りです。

表 5.4 分岐命令ごとの分岐距離指定子

命令		.S	.B	.W	.A	.L
BCnd	(Cnd=EQ/Z)	○	○	○	×	×
	(Cnd=NE/NZ)	○	○	○	×	×
	(Cnd= 上記以外)	×	○	×	×	×
BRA		○	○	○	○	○
BSR		×	×	○	○	○

5.1.5 オペランド部の記述方法

(1) 数値
プログラムに記述できる数値の種類は以下 5 つがあります。
記述した値は 32 ビット符号付きで処理されます。(浮動小数点数を除く)

(a) **2** 進数
0 ～ 1 のいずれかの数字で記述し、接尾辞として B または b を添付します。

- 記述例

```
1011000B
1011000b
```

(b) **8** 進数
0 ～ 7 までの数字で記述し、接尾辞として O または o を添付します。

- 記述例

```
60702O
60702o
```

(c) **10** 進数
0 ～ 9 までの数字で記述します。

- 記述例

```
9243
```

(d) **16** 進数
0 ～ 9, A ～ F, a ～ f で記述し、接尾辞として H または h を添付します。
アルファベットで始まる数値の場合は接頭辞として 0 を添付します。

- 記述例

```
0A5FH
5FH
0a5fh
5fh
```

(e) 浮動小数点数
浮動小数点数は制御命令 ".FLOAT" と ".DOUBLE" のオペランドにのみ記述できます。
浮動小数点数は式に記述できません。
浮動小数点数で表される次の範囲の値を記述できます。
FLOAT (32bits)1.17549435×10^{-38} ～ 3.40282347×10^{38}
DOUBLE (64bits)$2.2250738585072014\times10^{-308}$ ～ $1.7976931348623157\times10^{308}$

- 記述方法

```
( 仮数部 )E( 指数部 )
( 仮数部 )e( 指数部 )
```

- 記述例

```
3.4E35    ;3.4x10**35
3.4e-35   ;3.4x10**-35
-.5E20    ;-0.5x10**20
5e-20     ;5.0x10**-20
```

(2) アドレッシングモード
命令のオペランド部に記述できるアドレッシングモードは以下 3 つがあります。

(a) 一般命令アドレッシング

- レジスタ直接
指定したレジスタが演算の対象となります。R0 ～ R15、SP を記述することができます。-dpfpu 指定時はさらに DR0 ～ DR15、DRL0 ～ DRL15 及び DRH0 ～ DRH15 を記述することができます。SP は R0 と解釈します。(R0=SP)

```
Rn (Rn=R0 ～ R15, SP)
```

- 記述例

```
ADD  R1, R2
```

- 即値
#imm で示した即値は整数を表します。
#uimm で示した即値は符号なし整数を表します。
#simm で示した即値は符号付き整数を表します。
#imm:n、#uimm:n、および #simm:n は、n ビット長の即値を表します。

```
#imm:8, #uimm:8, #simm:8, #imm:16, #simm:16, #simm:24, #imm:32
```

注 RTSD 命令の #uimm:8 は、確定値でなければなりません。

- 記述例

```
MOV.L  #-100, R2          ; #simm:8
```

- レジスタ間接
レジスタの値が演算対象の実効アドレスになります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。

```
[Rn] (Rn=R0 ～ R15, SP)
```

- 記述例

```
ADD  [R1], R2
```

- レジスタ相対
ディスプレースメント (dsp) を 32 ビットにゼロ拡張した値と、レジスタ値を加算した結果が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。dsp:n は、n ビット長のディスプレースメントを表します。

```
dsp:8[Rn], dsp:16[Rn] (Rn=R0 ～ R15,SP)
```

dsp の値は以下規則によりスケーリングされた値で指定します。アセンブラではスケーリング前の値に戻し、命令のビットパターンに埋め込みます。

表 5.5 **dsp** 値スケーリング規則

命令	規則
サイズ指定子をとる転送命令	サイズ指定子 .B/.W/.L に応じてそれぞれ、1/2/4 倍
サイズ拡張指定子をとる演算命令	サイズ拡張指定子 .B/.UB/.W/.UW/.L に応じてそれぞれ、1/1/2/2/4 倍
ビット操作命令	1 倍
上記以外	4 倍

- 記述例

```
ADD  400[R1], R2              ; dsp:8[Rn] (400/4 = 100)
```

サイズ指定子 W/L で 2/4 の倍数でない場合、アセンブル時確定値はアセンブラエラー
アセンブル時未確定値はリンク時エラー

(b) 拡張命令アドレッシング

- 短縮即値
#imm で示した即値が演算の対象となります。即値がアセンブル時確定値でない場合はエラー処理されます。

#imm:1
このアドレッシングは、DSP 機能命令（RACW）の src にのみ使用されます。1 または 2 を記述できます。

- 記述例

```
RACW  #1                      ; RACW  #imm:1
```

#imm:2
#imm で示した 2 ビット即値が演算の対象となります。このアドレッシングは、コプロセッサ命令 (MVFCP,MVTCP,OPECP) のコプロセッサ番号指定にのみ使用されます。

- 記述例

```
MVTCP  #3, R1, #4:16          ; MVTCP  #imm:2, Rn, #imm:16
```

#imm:3
#imm で示した 3 ビット即値が演算の対象となります。このアドレッシングは、ビット操作命令 (BCLR,BMCnd,BNOT,BSET,BTST) のビット番号指定に使用されます。

- 記述例

```
BSET   #7, R10                ; BSET   #imm:3, Rn
```

#imm:4
ADD,AND,CMP,MOV,MUL,OR,SUB 命令のソースに使用される場合は、#imm で示した 4 ビット即値を 32 ビットにゼロ拡張した結果が演算の対象となります。
MVTIPL 命令の割り込み優先レベル指定に使用される場合は、#imm で示した 4 ビット即値が演算の対象となります。

- 記述例

```
ADD   #15, R8                 ; ADD   #imm:4, Rn
```

#imm:5
#imm で示した 5 ビット即値が演算の対象となります。このアドレッシングは、ビット操作命令 (BCLR, BMCnd, BNOT, BSET, BTST) のビット番号指定、シフト命令 (SHAR,SHLL,SHLR) のシフト幅指定、および、ローテート命令 (ROTL,ROTR) のローテート幅指定にのみ使用されます。

- 記述例

```
BSET  #31, R10                ; BSET   #imm:5, Rn
```

- 短縮レジスタ相対
5 ビットディスプレースメント (dsp) を 32 ビットにゼロ拡張した値と、レジスタ値を加算した結果が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。
dsp の値はサイズ指定子 .B/.W/.L に応じて、それぞれ 1/2/4 倍した値で指定します。dsp の値がアセンブル時確定値でない場合、エラー処理をします。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

```
dsp:5[Rn] (Rn=R0 ～ R7, SP)
```

- 記述例

```
MOV.L  R3,124[R1]              ; dsp:5[Rn] (124/4 = 31)
```

*src/dest のレジスタも R0 ～ R7 でなくてはなりません。

- ポストインクリメントレジスタ間接
レジスタの値に、サイズ指定子 .B/.W/.L に応じて、それぞれ 1/2/4 を加算します。更新前のレジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

```
[Rn+] (Rn=R0 ～ R15, SP)
```

- 記述例

```
MOV.L [R3+],R1
```

- プリデクリメントレジスタ間接
レジスタの値に、サイズ指定子 .B/.W/.L に応じて、それぞれ 1/2/4 を減算します。更新後のレジスタの値が演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

```
[-Rn] (Rn=R0 ～ R15, SP)
```

- 記述例

```
MOV.L [-R3],R1
```

- インデックス付きレジスタ間接
インデックスレジスタ (Ri) の値をサイズ指定子 .B/.W/.L に応じて、それぞれ 1/2/4 倍した値とベースレジスタ (Rb) の値を加算した結果の下位 32 ビットが演算対象の実効アドレスとなります。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。このアドレッシングは、MOV,MOVU 命令にのみ使用されます。

```
[Ri,Rb] (Ri=R0 ～ R15, SP)  (Rb=R0 ～ R15,tfe SP)
```

- 記述例

```
MOV.L [R3,R1],R2
MOV.L R3, [R1,R2]
```

(c) 特定命令アドレッシング

- 制御レジスタ直接
指定した制御レジスタが演算の対象となります。
このアドレッシングは、MVTC,POPC,PUSHC,MVFC 命令にのみ使用されます。

```
PSW, FPSW, USP, ISP, INTB, EXTB, BPSW, BPC, FINTV, PC
```

-dpfpu 指定時はさらに DPUSHM,DPOPM,MVTDC,MVFDC 命令でも使用されます。

```
DPSW, DCMR, DECNT, DEPC
```

- 記述例

```
STC  PSW,R2
```

- PSW 直接
指定したフラグ、または、ビットが演算の対象となります。このアドレッシングは、CLRPSW,SETPSW 命令にのみ使用されます。

```
U, I, O, S, Z, C
```

- 記述例

```
CLRPSW  U
```

- プログラムカウンタ相対
分岐命令の分岐先を指定するためのアドレッシング。

```
Rn(Rn=R0 ～ R15, SP)
```

プログラムカウンタの値と、Rn の値を符号付きで加算した結果が実効アドレスとなります。Rn の値の範囲は、-2147483648 ～ 2147483647 です。実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。このアドレッシングモードは、BRA(.L)、BSR(.L) 命令に使用されます。

```
label(PC + pcdsp:3)
```

分岐命令の分岐先アドレスを表します。指定したシンボル、数値が実効アドレスとなります。
指定した分岐先アドレスからプログラムカウンタの値を引いたものをディスプレースメント (pcdsp) として命令のビットパターンに埋め込みます。
分岐距離指定子が “.S” の場合、プログラムカウンタの値とディスプレースメントの値を符号なしで加算した結果の下位 32 ビットが実効アドレスとなります。
pcdsp の範囲は、3 ≦ pcdsp:3 ≦ 10 です。
実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。このアドレッシングは、
BRA,BCnd(Cnd==EQ,NE,Z,NZ のみ) のみに使用できます。

```
label(PC + pcdsp:8/pcdsp:16/pcdsp:24)
```

分岐命令の分岐先アドレスを表します。指定したシンボル、数値が実効アドレスとなります。

指定した分岐先アドレスからプログラムカウンタの値を引いたものをディスプレースメント (pcdsp) として命令のビットパターンに埋め込みます。
分岐距離指定子が “.B” または “.W” または “.A” の場合、プログラムカウンタの値とディスプレースメントの値を符号付きで加算した結果の下位 32 ビットが実効アドレスとなります。
pcdsp の範囲は以下の通りです。
“.B” の場合 128 ≦ pcdsp:8 ≦ +127
“.W” の場合 32768 ≦ pcdsp:16 ≦ +32767
“.A” の場合 8388608 ≦ pcdsp:24 ≦ +8388607
実効アドレスの範囲は、00000000h ～ FFFFFFFFh です。

(3) ビット長指定子
ビット長指定子はオペランドの即値、またはディスプレースメントのサイズを指定します。

- 記述方法

```
:width
```

- 内容
本指定子はオペランドに記述された即値、またはディスプレースメントの直後に指定します。
アセンブラは指定されたビット長のアドレッシングモードを選択します。
本指定子を省略した場合はアセンブラがもっとも効率のよいビット長を選択します。
本指定子が記述されている場合には最適選択は行わず、指定されたビット長とします。
本指定子はアセンブラ制御命令のオペランドには記述できません。
即値、ディスプレースメントの式と本指定子の間には、１つ以上の空白文字を入れることができます。
命令フォーマットに存在しないビット長が指定された場合は、エラー処理されます。
width に指定できるものは次の通りです。
2: 有効ビット長が 2 ビットであることを表します。

```
#imm:2
```

3: 有効ビット長が 3 ビットであることを表します。

```
#imm:3
```

4: 有効ビット長が 4 ビットであることを表します。

```
#imm:4
```

5: 有効ビット長が 5 ビットであることを表します。

```
#imm:5, dsp:5
```

8: 有効ビット長が 8 ビットであることを表します。

```
#uimm:8, #simm:8, dsp:8
```

16: 有効ビット長が 16 ビットであることを表します。

```
#uimm:16, #simm:16, dsp:16
```

24: 有効ビット長が 24 ビットであることを表します。

```
#simm:24
```

32: 有効ビット長が 32 ビットであることを表します。

```
#imm:32
```

(4) サイズ拡張指定子
サイズ拡張指定子は、演算命令でソースがメモリオペランドの場合、メモリオペランドのサイズと拡張方法を指定するために付加されます。

- 記述方法

```
.memex
```

- 内容
本指定子はメモリオペランドの直後に記述し、間に空白文字を入れることはできません。
サイズ拡張指定子は特定の命令と、メモリオペランドの組み合わせに対してのみ有効で、有効でない命令とオペランドの組み合わせで指定した場合は、エラー処理をします。
指定可能な命令とオペランドの組み合わせは、RX ファミリソフトウェアマニュアルの命令フォーマットのオペランドに .memex が付いているパターンのみです。
省略時はビット操作命令では 'B' として扱い、それ以外の命令では 'L' として扱います。
指定可能なサイズ拡張指定子と効果を、以下表に記します。

表 5.6 サイズ拡張指定子

サイズ拡張指定子	効果
B	8 ビットデータを 32 ビット符号拡張
UB	8 ビットデータを 32 ビットゼロ拡張
W	16 ビットデータを 32 ビット符号拡張
UW	16 ビットデータを 32 ビットゼロ拡張
L	32 ビットデータをロード

(記述例)
```
ADD  [R1].B, R2
AND  125[R1].UB, R2
```

5.1.6 式

数値、シンボルおよび演算子を組み合わせた式を記述できます。

- 演算子と数値の間には空白文字またはタブを記述できます。
- 演算子は複数組み合わせて記述できます。
- シンボル値として式を記述する場合は、式の値がアセンブル時に確定するように式を記述する必要があります。
- 式の項に文字定数は使用できません。
- 演算結果の範囲は、-2147483648 ～ 2147483647 となります。演算結果がこの範囲を超えた場合のオーバフローの判断は行いません。

(a) 演算子
プログラムに記述できる演算子の一覧を示します。

- 単項演算子

表 5.7 単項演算子

演算子	機能
+	続く値を正の値として扱います。
-	続く値を負の値として扱います。
~	続く値の論理否定値を扱います。
SIZEOF	オペランドに指定したセクションのサイズ（バイト数）を値として扱います。
TOPOF	オペランドに指定したセクションの開始アドレス値として扱います。

SIZEOF, TOPOF は、オペランドとの間に空白文字またはタブを記述します。
(例) SIZEOF program

- 二項演算子

表 5.8 二項演算子

演算子	機能
+	左辺値と右辺値を加算します。
-	左辺値から右辺値を減算します。
*	左辺値と右辺値を乗算します。
/	左辺値を右辺値で除算します。
%	左辺値を右辺値で割った余りを扱います。
>>	左辺値を右辺値回右へビットシフトします。
<<	左辺値を右辺値回左へビットシフトします。
&	左辺値と右辺値のビット毎の論理積値を扱います。
|	左辺値と右辺値のビット毎の論理和値を扱います。
^	左辺値と右辺値のビット毎の排他的論理和値を扱います。

- 条件演算子
  条件演算子は制御命令 ".IF", ".ELIF" のオペランドにだけ記述できます。

表 5.9 条件演算子

演算子	機能
>	左辺値が右辺値より大きいことを評価します。
<	左辺値が右辺値より小さいことを評価します。
>=	左辺値が右辺値以上であることを評価します。
<=	左辺値が右辺値以下であることを評価します。
==	左辺値が右辺値と等しいことを評価します。
!=	左辺値が右辺値と等しくないことを評価します。

- 演算優先順位変更演算子

表 5.10 演算優先順位変更演算子

演算子	機能
()	() で囲った演算を最優先で行います。一つの式に複数の () が記述されている場合は、左側が優先されます。() はネストした記述ができます。

(b) 式の演算優先順位
オペランドに記述されている式について、次に示す優先順位に従い演算を行った結果の数値を値として扱います。

- 演算子がもつ優先順位の高いものから演算します。演算子の優先順位を以下表に示します。
- 同一の優先順位を持つ演算子は、左から順に演算を行います。
- () で囲ったものが、優先順位が一番高くなります。

表 5.11 式の演算優先順位

優先順位	演算子の種類	演算子
1	演算優先順位変更演算子	()

優先順位	演算子の種類	演算子
2	単項演算子	+, -, ~, SIZEOF, TOPOF
3	二項演算子 1	*, /, %
4	二項演算子 2	+, -
5	二項演算子 3	>>, <<
6	二項演算子 4	&
7	二項演算子 5	|, ^
8	条件演算子	>, <, >=, <=, ==, !=

5.1.7 コメントの記述方法

セミコロン (;) の後に続けて記述します。セミコロンから行末までをコメントと見なします。

- 記述例

```
ADD     R1, R2          ; R2 に R1 を加えます。
```

5.1.8 命令フォーマットの最適選択

RX ファミリの命令には、同一処理に対して複数の命令フォーマットを持つものがあります。
アセンブラでは、命令およびアドレッシングモードの指定に応じて、最短コードの命令フォーマットを選択する最適選択を行います。

(1) 即値について
アセンブラではオペランドに即値を持つ命令である場合、オペランドに指定された即値の範囲に従い選択可能なアドレッシングから最適選択を行います。以下に即値の範囲について優先順位の高い順に示します。

表 5.12 即値の範囲

#imm	10 進記法	16 進記法
#imm:1	1 ～ 2	1H ～ 2H
#imm:2	0 ～ 3	0H ～ 3H
#imm:3	0 ～ 7	0H ～ 7H
#imm:4	0 ～ 15	0H ～ 0FH
#imm:5	0 ～ 31	0H ～ 1FH
#imm:8	-128 ～ 255	-80H ～ 0FFH
#uimm:8	0 ～ 255	0H ～ 0FFH
#simm:8	-128 ～ 127	-80H ～ 7FH
#imm:16	-32768 ～ 65535	-8000H ～ 0FFFFH
#simm:16	-32768 ～ 32767	-8000H ～ 7FFFH
#simm:24	-8388608 ～ 8388607	-800000H ～ 7FFFFFH
#imm:32	-2147483648 ～ 4294967295	-80000000H ～ 0FFFFFFFFH

注 1. 16 進表記は 32 ビット表記も可能です。
例：10 進表記 ”-127”、16 進表記 ”-7FH” は ”0FFFFFF81H” と表記できます。

注 2. INT 命令の src の #imm の範囲は 0 ～ 255 となります。

注 3. RTSD 命令の src の #imm の範囲は #uimm:8 を 4 倍した値となります。

(2) **ADC, SBB** 命令

ADC, SBB 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

注 最適選択対象とならない命令フォーマットとオペランドは記述していません。表内の処理サイズは、特に明記がない場合は "L" となります。

表 5.13 **ADC, SBB** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
ADC src,dest	#simm:8 #simm:16 #simm:24 #imm:32	- - - -	Rd Rd Rd Rd	4 5 6 7
ADC/SBB src,dest	dsp:8[Rs].L dsp:16[Rs].L	- -	Rd Rd	4 5

SBB 命令では、src に即値を指定することはできません。

(3) **ADD** 命令

ADD 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.14 **ADD** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
(1) ADD src,dest	#uimm:4 #simm:8 #simm:16 #simm:24 #imm:32	- - - - -	Rd Rd Rd Rd Rd	2 3 4 5 6
	dsp:8[Rs].memex dsp:16[Rs].memex	- -	Rd Rd	3(memex =UB), 4(memex ≠ UB) 4(memex =UB), 5(memex ≠ UB)
(2) ADD src,src2,dest	#simm:8 #simm:16 #simm:24 #imm:32	Rs Rs Rs Rs	Rd Rd Rd Rd	3 4 5 6

(4) **AND, OR, SUB, MUL** 命令

AND, OR, SUB, MUL 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.15 **AND, OR, SUB** および **MUL** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
AND/OR/SUB/MUL src,dest	#uimm:4 #simm:8 #simm:16 #simm:24 #imm:32	- - - - -	Rd Rd Rd Rd Rd	2 3 4 5 6
	dsp:8[Rs].memex dsp:16[Rs].memex	- -	Rd Rd	3(memex = UB), 4(memex ≠ UB) 4(memex = UB), 5(memex ≠ UB)

SUB 命令では、src に #simm:8/16/24, #imm32 を指定することはできません。

(5) **BMCnd** 命令
BMCnd 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.16 **BMCnd** 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ [バイト]
		src	src2	dest	
BMCnd src,dest	B B	#imm:3 #imm:3	- -	dsp:8[Rs].B dsp:16[Rs].B	4 5

(6) **CMP** 命令
CMP 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.17 **CMP** 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ [バイト]
		src	src2	dest	
CMP src,src2	L L L L L L	#uimm:4 #uimm:8 #simm:8 #simm:16 #simm:24 #imm:32	Rd Rd Rd Rd Rd Rd	- - - - - -	2 3 3 4 5 6
	L L	dsp:8[Rs].memex dsp:16[Rs].memex	Rd Rd	- -	3(memex = UB), 4(memex ≠ UB) 4(memex = UB), 5(memex ≠UB)

(7) **DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, TST, XOR** 命令
DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, MUL, TST, XOR 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.18 **DIV, DIVU, EMUL, EMULU, ITOF, MAX, MIN, TST** および **XOR** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
DIV/DIVU/ EMUL/EMULU/ITOF/ MAX/MIN/TST/XOR	#simm:8 #simm:16 #simm:24 #imm:32	- - - -	Rd Rd Rd Rd	4 5 6 7
src,dest	dsp:8[Rs].memex dsp:16[Rs].memex	- -	Rd Rd	4(memex=UB), 5(memex ≠ UB) 5(memex=UB), 6(memex ≠ UB)

ITOF 命令では、src に #simm:8/16/24, #imm32 を指定することはできません。

(8) **FADD, FCMP, FDIV, FMUL, FTOI** 命令
FADD, FCMP, FDIV, FMUL, FTOI 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.19 **FADD, FCMP, FDIV, FMUL** および **FTOI** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
FADD/FCMP/FDIV/ FMUL/FTOI src,dest	#imm:32 dsp:8[Rs].L dsp:16[Rs].L	- - -	Rd Rd Rd	7 4 5

FTOI 命令では、src に #imm32 を指定することはできません。

(9) **MVTC, STNZ, STZ** 命令

MVTC, STNZ, STZ 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.20 **MVTC, STNZ** および **STZ** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
MVTC/STNZ/STZ src,dest	#simm:8 #simm:16 #simm:24 #imm:32	- - - -	Rd Rd Rd Rd	4 5 6 7

(10) **MOV** 命令

MOV 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.21 **MOV** 命令の命令フォーマット

命令フォーマット	size	処理サイズ	対象			コードサイズ [バイト]
			src	src2	dest	
MOV(.size) src,dest	B/W/L	size	Rs(Rs=R0-R7)	-	dsp:5[Rd](Rd=R0-R7)	2
	B/W/L	L	dsp:5[Rs](Rs=R0-R7)	-	Rd(Rd=R0-R7)	2
	B/W/L	L	#uimm:8	-	dsp:5[Rd](Rd=R0-R7)	3
	L L L L L L	L L L L L L	#uimm:4 #uimm:8 #simm:8 #simm:16 #simm:24 #imm:32	- - - - - -	Rd Rd Rd Rd Rd Rd	2 3 3 4 5 6
	B W/L W L L L	B W/L W L L L	#imm:8 #simm:8 #imm:16 #simm:16 #simm:24 #imm:32	- - - - - -	[Rd] [Rd] [Rd] [Rd] [Rd] [Rd]	3 3 4 4 5 6
	B W/L W L L L	B W/L W L L L	#imm:8 #simm:8 #imm:16 #simm:16 #simm:24 #imm:32	- - - - - -	dsp:8[Rd] dsp:8[Rd] dsp:8[Rd] dsp:8[Rd] dsp:8[Rd] dsp:8[Rd]	4 4 5 5 6 7
	B W/L W L L L	B W/L W L L L	#imm:8 #simm:8 #imm:16 #simm:16 #simm:24 #imm:32	- - - - - -	dsp:16[Rd] dsp:16[Rd] dsp:16[Rd] dsp:16[Rd] dsp:16[Rd] dsp:16[Rd]	5 5 6 6 7 8
	B/W/L B/W/L	L L	dsp:8[Rs] dsp:16[Rs]	- -	Rd Rd	3 4
	B/W/L B/W/L	size size	Rs Rs	- -	dsp:8[Rd] dsp:16[Rd]	3 4
	B/W/L B/W/L	size size	[Rs] [Rs]	- -	dsp:8[Rd] dsp:16[Rd]	3 4
	B/W/L B/W/L	size size	dsp:8[Rs] dsp:16[Rs]	- -	[Rd] [Rd]	3 4
	B/W/L B/W/L	size size	dsp:8[Rs] dsp:8[Rs]	- -	dsp:8[Rd] dsp:16[Rd]	4 5
	B/W/L B/W/L	size size	dsp:16[Rs] dsp:16[Rs]	- -	dsp:8[Rd] dsp:16[Rd]	5 6

(11) **MOVU** 命令
MOVU 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.22 **MOVU** 命令の命令フォーマット

命令フォーマット	size	処理サイズ	対象			コードサイズ [バイト]
			src	src2	dest	
MOVU(.size) src,dest	B/W	L	dsp:5[Rs](Rs=R0-R7)	-	Rd(Rd=R0-R7)	2
	B/W B/W	L L	dsp:8[Rs] dsp:16[Rs]	- -	Rd Rd	3 4

(12) **PUSH** 命令
PUSH 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.23 **PUSH** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
PUSH src	dsp:8[Rs] dsp:16[Rs]	- -	- -	3 4

(13) **ROUND** 命令
ROUND 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.24 **ROUND** 命令の命令フォーマット

命令フォーマット	対象			コードサイズ [バイト]
	src	src2	dest	
ROUND src,dest	dsp:8[Rs] dsp:16[Rs]	- -	Rd Rd	4 5

(14) **SCCnd** 命令
SCCnd 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.25 **SCCnd** 命令の命令フォーマット

命令フォーマット	size	対象			コードサイズ [バイト]
		src	src2	dest	
SCCnd(.size) src,dest	B/W/L B/W/L	- -	- -	dsp:8[Rd] dsp:16[Rd]	4 5

(15) **XCHG** 命令
XCHG 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.26 **XCHG** 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ [バイト]
		src	src2	dest	
XCHG src,dest	L L	dsp:8[Rs].memex dsp:16[Rs].memex	- -	Rd Rd	4(memex = UB), 5(memex ≠ UB) 5(memex = UB), 6(memex ≠ UB)

(16) **BCLR, BNOT, BSET, BTST** 命令
BCLR, BNOT, BSET, BTST 命令に対して、最適選択の対象となる命令フォーマットとオペランドを優先順位の高い順に示します。

表 5.27 **BCLR, BNOT, BSET** および **BTST** 命令の命令フォーマット

命令フォーマット	処理サイズ	対象			コードサイズ [バイト]
		src	src2	dest	
BCLR/BNOT/BSET/ BTST src,dest	B B	#imm:3 #imm:3	- -	dsp:8[Rd].B dsp:16[Rd].B	3 4
	B B	Rs Rs	- -	dsp:8[Rd].B dsp:16[Rd].B	4 5

5.1.9 分岐命令の最適選択

(1) 相対無条件分岐命令 **(BRA)** の最適選択

(a) 指定可能な分岐距離指定子
.S 3 ビット PC 相対 (PC+pcdsp:3,3 ≦ pcdsp:3 ≦ 10)
.B 8 ビット PC 相対 (PC+pcdsp:8,-128 ≦ pcdsp:8 ≦ 127)
.W 16 ビット PC 相対 (PC+pcdsp:16,-32768 ≦ pcdsp:16 ≦ 32767)
.A 24 ビット PC 相対 (PC+pcdsp:24,-8388608 ≦ pcdsp:24 ≦ 8388607)
.L レジスタ相対 (PC+Rs,-2147483648 ≦ Rs ≦ 2147483647)
* レジスタ相対はオペランドがレジスタの場合のみ選択され、最適選択によって選択されることはありません。

(b) 最適選択

- アセンブラでは相対無条件分岐命令のオペランドが分岐最適化対象条件を満たす場合、最短の分岐距離を選択します。条件については、「5.1.4 (3) 分岐距離指定子」を参照してください。
- 条件を満たさないものについては、24 ビット PC 相対 (.A) を選択します。

(2) 相対サブルーチン分岐命令 **(BSR)** の最適選択

(a) 指定可能な分岐距離指定子
.W 16 ビット PC 相対 (PC+pcdsp:16,-32768 ≦ pcdsp:16 ≦ 32767)
.A 24 ビット PC 相対 (PC+pcdsp:24,-8388608 ≦ pcdsp:24 ≦ 8388607)
.L レジスタ相対 (PC+Rs,-2147483648 ≦ Rs ≦ 2147483647)
* レジスタ相対はオペランドがレジスタの場合のみ選択され、最適選択によって選択されることはありません。

(b) 最適選択

- アセンブラでは相対サブルーチン分岐命令のオペランドが分岐最適化対象条件を満たす場合、最短の分岐距離を選択します。条件については、「5.1.4 (3) 分岐距離指定子」を参照してください。
- 条件を満たさないものについては、24 ビット PC 相対 (.A) を選択します。

(3) 条件分岐命令 **(BCnd)** の最適選択

(a) 指定可能な分岐距離指定子
BEQ.S 3 ビット PC 相対 (PC+pcdsp:3,3 ≦ pcdsp:3 ≦ 10)
BNE.S 3 ビット PC 相対 (PC+pcdsp:3,3 ≦ pcdsp:3 ≦ 10)
BCnd.B 8 ビット PC 相対 (PC+pcdsp:8,-128 ≦ pcdsp:8 ≦ 127)
BEQ.W 16 ビット PC 相対 (PC+pcdsp:16,-32768 ≦ pcdsp:16 ≦ 32767)
BNE.W 16 ビット PC 相対 (PC+pcdsp:16,-32768 ≦ pcdsp:16 ≦ 32767)

(b) 最適選択

- アセンブラでは条件分岐命令のオペランドが分岐最適化対象条件を満たす場合、論理を反転した条件分岐命令と最適な分岐距離の相対無条件分岐命令を組み合わせた最適な条件分岐コードを選択して生成します。
- 条件を満たさないものについては、8 ビット PC 相対 (.B) または 16 ビット PC 相対 (.W) を選択します。

(c) 変換される条件分岐命令に対する代替命令

表 5.28 条件分岐命令の代替規則

条件分岐命令	代替分岐命令	条件分岐命令	代替分岐命令
BNC/BLTU dest	BC ..xx BRA.A dest ..xx:	BC/BGEU dest	BNC ..xx BRA.A dest ..xx:
BLEU dest	BGTU ..xx BRA.A dest ..xx:	BGTU dest	BLEU ..xx BRA.A dest ..xx:
BNZ/BNE dest	BZ ..xx BRA.A dest ..xx:	BZ/BEQ dest	BNZ ..xx BRA.A dest ..xx:
BPZ dest	BN ..xx BRA.A dest ..xx:	BO dest	BNO ..xx BRA.A dest ..xx:
BGT dest	BLE ..xx BRA.A dest ..xx:	BLE dest	BGT ..xx BRA.A dest ..xx:
BGE dest	BLT ..xx BRA.A dest ..xx:	BLT dest	BGE ..xx BRA.A dest ..xx:

上記は相対無条件分岐命令の分岐距離が 24 ビット PC 相対の場合を記します。
“..xx” ラベルおよび相対無条件分岐命令は内部的に処理されるものであり、アセンブルリストファイルにはコードのみ生成されます。

5.1.10 代用レジスタ名【PID 機能向け】

次の代用レジスタ名を汎用レジスタ名の代わりに使用することができます。

表 5.29 即値の範囲

代用レジスタ名	対応する汎用レジスタ名
__PID_R0	R0
__PID_R1	R1
__PID_R2	R2
__PID_R3	R3
__PID_R4	R4
__PID_R5	R5
__PID_R6	R6
__PID_R7	R7
__PID_R8	R8
__PID_R9	R9
__PID_R10	R10
__PID_R11	R11
__PID_R12	R12

代用レジスタ名	対応する汎用レジスタ名
__PID_R13	R13
__PID_R14	R14
__PID_R15	R15
__PID_REG	PID として選択されるレジスタ 注 1

注 1. -pid または -nouse_pid_register オプションを選択した場合に、PID レジスタとして選択されるレジスタです。
PID レジスタの選択ルールについては、アセンブル・オプションの -pid および -nouse_pid_register オプションの項目を参照してください。

PID 機能のマスタープログラムを構成するアセンブリソース上では、PID レジスタに選択される可能性のあるレジスタは、汎用レジスタ名 (R13 など) ではなく、代用レジスタ名で記述してください。
代用レジスタ名は、それが PID レジスタに選択されているときに、nouse_pid_register を有効にしてアセンブルしてもエラーにはなりません。

［備考］
- 代用レジスタ名は、-nouse_pid_register および -pid オプションが選択されていない場合でも使用できます。

5.2 擬似命令

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、オブジェクト・コード（機械語）に変換されますが、疑似命令は、原則としてオブジェクト・コードに変換されません。
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、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。

種類	擬似命令
リンク制御擬似命令	.SECTION, .GLB, .RVECTOR
アセンブル制御擬似命令	.EQU, .END, .INCLUDE
アドレス制御擬似命令	.ORG, .OFFSET, .ENDIAN, .BLKB, .BLKW, .BLKL, .BLKD, .BYTE, .WORD, .LWORD, .FLOAT, .DOUBLE, .ALIGN
マクロ制御擬似命令	.MACRO, .EXITM, .LOCAL, .ENDM, .MREPEAT, .ENDR, ..MACPARA, ..MACREP, .LEN, .INSTR, .SUBSTR
コンパイラ専用制御擬似命令	._LINE_TOP, ._LINE_END, .SWSECTION, .SWMOV, .SWITCH, .INSTALIGN

以降、各疑似命令について詳細な説明を行います。

5.2.2 リンク制御擬似命令

プログラムを複数のファイルに分割して記述するリロケータブルアセンブルを実行するための制御命令です。

.SECTION

セクションの宣言、再開を指定します。

[指定形式]

```
.SECTION △ < セクション名 >
.SECTION △ < セクション名 >,< セクション属性 >
.SECTION △ < セクション名 >,< セクション属性 >,ALIGN=[2|4|8]
.SECTION △ < セクション名 >,ALIGN=[2|4|8]
< セクション属性 >:[CODE|ROMDATA|DATA]
```

[詳細説明]
セクションの宣言、再開を指定します。
(1) セクションの宣言
セクション名、セクション属性を指定し、セクションの始まりを定義します。
(2) セクションの再開
ソースプログラム中にすでに存在しているセクションを再開します。セクションの再開ではすでに存在するセクション名を指定します。セクション属性とアライメント補正値は最初に宣言したものを継続します。
'ALIGN=' 指定がある場合、指定されたセクションに対してアライメント補正値を変更することができます。
ALIGN 指定をした相対アドレス形式セクションまたは絶対アドレス形式セクションに、制御命令 ".ALIGN" が記述できます。
ALIGN 指定がない場合、そのセクションの境界調整数は 1 となります。

例

```
.SECTION P,CODE
 NOP

.SECTION B_1,DATA
.BLKB    10

.SECTION D_1,ROMDATA
.BYTE    "abcd"
.SECTION  D,ROMDATA,ALIGN=4
.LWORD    11111111H,22222222H

.END
```

[備考]
セクション名は必ず記述してください。
メモリ領域を確保したりメモリにデータを格納するアセンブラ制御命令を記述する場合は、必ず本制御命令でセクションを定義してください。
ニーモニックを記述する場合は必ず、本制御命令でセクションを定義してください。
セクション属性と ALIGN は、セクション名の後に記述してください。
セクション属性および、ALIGN 指定をする場合は、カンマで区切って記述してください。
セクション属性と ALIGN の記述順序は任意です。
セクション属性は、'CODE','ROMDATA','DATA' のいずれかを記述できます。
セクション属性は省略できます。このとき、アセンブラはセクション属性を CODE として処理します。
-endian=big 指定時、絶対アドレス形式の CODE セクションの開始アドレスには 4 の倍数以外の値を指定することはできません。
-endian=big 指定時、絶対アドレス形式の CODE セクションはウォーニングを出力し、セクションサイズが 4 の倍数になるようにアセンブラがセクション末尾に NOP(0x03) を書き込みます。
定義されているセクション名と同じ名前のシンボルを定義することはできません。セクションとシンボルを同じ名前で定義した場合、後で定義したものは A2118 エラーとなります。
$iop という名前のセクション名を定義することはできません。$iop を定義した場合 A2049 エラーとなります。

.GLB

本擬似命令で指定したラベルおよびシンボルがグローバルであることを宣言します。

[指定形式]

```
.GLB △ < 名前 >
.GLB △ < 名前 >[,< 名前 > …]
```

[詳細説明]
本制御命令で指定したラベルおよびシンボルがグローバルであることを宣言します。
本制御命令で指定したラベルおよびシンボルで、ファイル内で定義されていないものは、外部のファイルで定義されているものとして処理します。
本制御命令で指定したラベルおよび、シンボルで、ファイル内で定義されているものは、外部から参照できるように処理します。

例
```
  .GLB    name1,name2,name3
  .GLB    name4
  .SECTION P,code
   MOV.L  #name1,R1
```

[備考]
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドにグローバルラベルとするラベル名を記述します。
オペランドにグローバルシンボルとするシンボル名を記述します。
オペランドに複数のシンボル名を記述する場合は、カンマ (,) で区切って記述してください。

.RVECTOR

本制御命令で指定したラベルおよびシンボルを、可変ベクタとして登録します。
[指定形式]

```
.RVECTOR △ < 番号 >,< 名前 >
```

[詳細説明]
本制御命令で指定したラベルおよびシンボルを、可変ベクタとして登録します。
本制御命令の < 番号 > には、ベクタ番号として 0 ～ 255 の定数を記述することができます。
本制御命令の < 名前 > には、ファイル内で定義されたラベルまたはシンボルを指定することができます。
登録された可変ベクタは、最適化リンケージエディタにより、ひとつの C$VECT セクションにまとめられます。
【V3.00.00 以降】最適化リンケージエディタに -split_vect オプションを指定した場合は、C$VECT セクションはベクタテーブル番号ごとに分割され、個々のセクションは C$VECT< ベクタテーブル番号 > という名前を持ちます。

例
```
   .RVECTOR 50,_rvfunc
_rvfunc:
   MOV.L   #0,R1
   RTE
```

[備考]
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。

5.2.3 アセンブル制御擬似命令

制御命令自身はデータを生成しません。命令に対する機械語コードの生成を制御する制御命令です。アドレスの更新は行いません。

.EQU

シンボルに 32 ビット符号付き整数値 (-2147483648 ～ 2147483647) の範囲の値を定義します。

[指定形式]

< 名前 > △ .EQU △ < 数値 >

[詳細説明]
シンボルに 32 ビット符号付き整数値 (-2147483648 ～ 2147483647) の範囲の値を定義します。
本制御命令でシンボルを定義することにより、シンボリックデバッグ機能が使用できます。

例
symbol .EQU 1
symbol1 .EQU symbol+symbol
symbol2 .EQU 2

[備考]
シンボルに定義できる値は、アセンブル時に確定しなければなりません。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
シンボル定義のオペランドには、シンボルを記述できます。ただし、前方参照となるシンボル名は記述できません。
オペランドには式を記述できます。
シンボルはグローバル指定ができます。
本制御命令と .DEFINE 制御命令で同名のシンボルを宣言した場合、先に記述した方が優先されます。

.END

アセンブリ言語ファイルの終了を宣言します。

[指定形式]

.END

[詳細説明]
アセンブリ言語ファイルの終了を宣言します。
本制御命令を記述した行以降の記述内容はアセンブルリストファイルに出力するのみで、コード生成などの処理は行いません。

例
.END

[備考]
本制御命令は、1 つのアセンブリ言語ファイルに必ず 1 つ記述する必要があります。

.INCLUDE

アセンブリ言語ファイルの行に、インクルードファイルの内容全てを読み込みます。

[指定形式]

.INCLUDE △ < インクルードファイル名 >

[詳細説明]
アセンブリ言語ファイルの行に、インクルードファイルの内容全てを読み込みます。
本制御命令で読み込まれたインクルードファイルの内容は、読み込んだアセンブリ言語
ファイル内に記述した場合と、同じ 1 つのアセンブリ言語ファイルとして処理されます。
インクルードファイルは 30 レベルまでネストできます。
インクルードファイル名に絶対パスを記述した場合は、記述したディレクトリ内のファイルを検索します。
ファイルが見つからない場合はエラーとなります。
インクルードファイル名に絶対パスを記述していない場合は、次に示す順序でファイルを検索します。
(1)アセンブラ起動時にコマンド行で指定したアセンブリ言語ファイル名にディレクトリ指定がない場合は、.INCLUDE 制御命令で指定されたインクルードファイル名を検索します。アセンブラ起動時にコマンド行で指定したアセンブリ言語ファイル名にディレクトリ指定がある場合は、.INCLUDE 制御命令で指定されたインクルードファイル名にコマンド行で指定されたディレクトリ名を付加して検索します。
(2)アセンブラオプション -include で指定されたディレクトリを検索します。

(3)環境変数 INC_RXA に設定されているディレクトリを検索します。
例
```
.INCLUDE initial.src
.INCLUDE ..FILE@.inc
```

[備考]
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドのインクルードファイル名には、必ずファイル拡張子を記述してください。
オペランドには、制御命令 "..FILE" や "@" を含む文字列が記述できます。
ファイル名は、その先頭を除き、空白文字を含むことができます。
ファイル名をダブルクォーテーション「"」で囲わないでください。
自分自身をインクルードファイルに指定することはできません。

5.2.4 アドレス制御擬似命令

アセンブラがアドレス更新をする場合の指示を行う制御命令です。
絶対アドレス形式セクション内のアドレスを除いて、アセンブラが制御を行うアドレスはリロケータブル値となります。

.ORG

本制御命令を記述したセクションを絶対アドレス形式とします。

[指定形式]

```
.ORG △ < 数値 >
```

[詳細説明]
本制御命令を記述したセクションを絶対アドレス形式とします。
本制御命令を記述したセクションのアドレスはアブソリュート値になります。
本制御命令を記述した直後の行から記述したニーモニックのコードが格納されるアドレスを決定します。
本制御命令の直後の行から記述した領域確保制御命令で、確保されるメモリのアドレスを決定します。

例
```
.SECTION  D_1,ROMDATA
.ORG      0FF00H
.BYTE     "abcdefghijklmnopqrstuvwxyz"
.ORG      0FF80H
.BYTE     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

以下の場合は .SECTION の直後に .ORG が記述されていないため、エラーとなります。
```
.SECTION  D_1,ROMDATA
.BYTE     "abcdefghijklmnopqrstuvwxyz"
.ORG      0FF80H
.BYTE     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
.END
```

[備考]
本制御命令は、必ずセクション制御命令の直後に記述してください。
".SECTION" を記述した直後の行に ".ORG" の記述がない場合は、そのセクションは相対アドレス形式セクションとなります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドに記述できる値は、0 ～ 0FFFFFFFFH の範囲の数値です。
オペランドには式、シンボルを記述できます。ただし、アセンブル時に確定する値でなければなりません。
本制御命令は、相対アドレス形式指定を行ったセクション内には記述できません。
絶対アドレス形式セクション内であれば複数回記述できます。ただし本制御命令記述行のアドレスよりも小さい値を指定した場合にはエラーとなります。
本制御命令は、記述した箇所に、アドレスを進めたバイト数だけ、0 個以上の無効を意味するデータ (03H) を埋め込みます。

.OFFSET

セクション先頭からのオフセットを指定します。

[指定形式]

```
.OFFSET △ < 数値 >
```

[詳細説明]
セクション先頭からのオフセットを指定します。
本制御命令を記述した直後の行から記述したニーモニックのコードが格納される、セクション先頭からのオフセットを決定します。
本制御命令の直後の行から記述した領域確保制御命令で確保されるメモリの、セクション先頭からのオフセットを決定します。

例
```
 .SECTION  D_1,ROMDATA
 .BYTE     "abcdefghijklmnopqrstuvwxyz"
 .OFFSET   80H
 .BYTE     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 .END
```

以下の場合は .OFFSET 記述行のオフセットよりも小さい値が指定されているため、エラーとなります。

```
 .SECTION  D_1,ROMDATA
 .OFFSET   80H
 .BYTE     "abcdefghijklmnopqrstuvwxyz"
 .OFFSET   70H
 .BYTE     "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 .END
```

[備考]
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドに記述できる値は、0 ～ 0FFFFFFFFH の範囲の数値です。
オペランドには式、シンボルを記述できます。ただし、アセンブル時に確定する値でなければなりません。
本制御命令は、絶対アドレス形式指定を行ったセクション内には記述できません。
相対アドレス形式セクション内であれば複数回記述できます。ただし本制御命令記述行のオフセットよりも小さい値を指定した場合にはエラーとなります。
本制御命令は、記述した箇所に、アドレスを進めたバイト数だけ、0 個以上の無効を意味するデータ (03H) を埋め込みます。

.ENDIAN

本制御命令を記述したセクションのエンディアンを指定します。

[指定形式]

```
.ENDIAN △ BIG
.ENDIAN △ LITTLE
```

[詳細説明]
本制御命令を記述したセクションのエンディアンを指定します。
.ENDIAN BIG を記述したセクションのデータのバイト並びは Big Endian になります。
.ENDIAN LITTLE を記述したセクションのデータのバイト並びは Little Endian になります。
本制御命令が記述されていないセクションのデータのバイト並びは、-endian オプションに依存します。

例
```
 .SECTION  DT_B,ROMDATA
 .ORG      0FF00H
 .ENDIAN   BIG
 .LWORD    012345678H
```

```
.END
```

以下の場合は .SECTION または .ORG の直後に .ENDIAN が記述されていないため、エラーとなります。

```
.SECTION  DT_B,ROMDATA
.ORG      0FF00H
.LWORD    012345678H
.ENDIAN   BIG
.LWORD    0ABCDEF90H
.END
```

[備考]
本制御命令は必ず .SECTION 制御命令、またはそれに続く .ORG 制御命令の直後に記述してください。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
セクション属性が CODE のセクションに本制御命令を追加することはできません。

.BLKB

1 バイト単位で、指定したバイト数の RAM 領域を確保します。

[指定形式]

△ .BLKB △ < オペランド >
△ < ラベル名 :> △ .BLKB △ < オペランド >

[詳細説明]
1 バイト単位で、指定したバイト数の RAM 領域を確保します。
確保した RAM のアドレスに、ラベル名を定義することもできます。

例

```
symbol .EQU  1
       .SECTION B_1,DATA
work1: .BLKB 1
work2: .BLKB symbol
       .BLKB symbol+1
       .END
```

[備考]
本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",DATA" を記述することでセクション属性が DATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドの値は、アセンブル時に確定しなければなりません。
領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。
ラベル名には、必ずコロン (:) を記述してください。
オペランドに指定できる値の最大値は 7FFFFFFFH です。

.BLKW

2 バイト単位で、指定した個数の RAM 領域を確保します。

[指定形式]

△ .BLKW △ < オペランド >
△ < ラベル名 :> △ .BLKW △ < オペランド >

[詳細機能]
2 バイト単位で、指定した個数の RAM 領域を確保します。
確保した RAM のアドレスに、ラベル名を定義することもできます。

例
```
symbol .EQU 1
      .SECTION B_2,DATA,ALIGN=2
work1: .BLKW 1
work2: .BLKW symbol
      .BLKW symbol+1
      .END
```

[備考]
本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",DATA" を記述することでセクション属性が DATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドの値は、アセンブル時に確定しなければなりません。
領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。
ラベル名には、必ずコロン (:) を記述してください。
オペランドに指定できる値の最大値は 3FFFFFFFH です。

.BLKL

4 バイト単位で、指定した個数の RAM 領域を確保します。

[指定形式]

△ .BLKL △ < オペランド >
△ < ラベル名 :> △ .BLKL △ < オペランド >

[詳細説明]
4 バイト単位で、指定した個数の RAM 領域を確保します。
確保した RAM のアドレスに、ラベル名を定義することもできます。

例
```
symbol .EQU 1
      .SECTION B,DATA,ALIGN=4
work1: .BLKL 1
work2: .BLKL symbol
      .BLKL symbol+1
      .END
```

[備考]
本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",DATA" を記述することでセクション属性が DATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドの値は、アセンブル時に確定しなければなりません。
領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。
ラベル名には、必ずコロン (:) を記述してください。
オペランドに指定できる値の最大値は 1FFFFFFFH です。

.BLKD

8 バイト単位で、指定した個数の RAM 領域を確保します。

[指定形式]

△ .BLKD △ < オペランド >
△ < ラベル名 :> △ .BLKD △ < オペランド >

[詳細説明]
8 バイト単位で、指定した個数の RAM 領域を確保します。
確保した RAM のアドレスに、ラベル名を定義することもできます。

例
```
symbol .EQU 1
       .SECTION B,DATA,ALIGN=4
work1: .BLKD 1
work2: .BLKD symbol
       .BLKD symbol+1
       .END
```

[備考]
本制御命令は必ず、DATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",DATA" を記述することでセクション属性が DATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドの値は、アセンブル時に確定しなければなりません。
領域にラベル名を定義する場合は、制御命令の前にラベルを記述してください。
ラベル名には、必ずコロン (:) を記述してください。
オペランドに指定できる値の最大値は 0FFFFFFFH です。

.BYTE

1 バイト長の固定データを ROM に格納します。

[指定形式]

△ .BYTE △ < オペランド >
△ < ラベル名 :> △ .BYTE △ < オペランド >

[詳細説明]
1 バイト長の固定データを ROM に格納します。
データを格納したアドレスに、ラベル名を定義することもできます。

例
```
<endian=little オプション指定時 >
   .SECTION D_1,ROMDATA
symbol:
   .BYTE   1
   .BYTE   "data"
   .BYTE   symbol
   .BYTE   symbol+1
   .BYTE   1,2,3,4,5
   .END
<endian=big オプション指定時 >
   .SECTION PB,CODE,ALIGN=4
   MOV.L   R1,R2
   .ALIGN  4
   .BYTE   080H,00H,00H,00H
   .END
```

[備考]
本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",ROMDATA" を記述することでセクション属性が ROMDATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドに文字・文字列を記述するときは、クォーテーション (') またはダブルクォーテーション (") で囲ってください。このとき格納されるデータは、文字の ASCII コードになります。
ラベルを定義する場合には、制御命令の前にラベル名を記述してください。
ラベル名には、必ずコロン (:) を記述してください。
endian=big オプション指定時、本制御命令は次の条件に当てはまるセクション内にのみ記述できます。条件に当てはまらないセクション内に記述した場合はエラーとなります。
(1) ROMDATA セクション
```
.SECTION data,ROMDATA
```
(2) セクション定義の際のアドレス補正に 4、もしくは 8 を指示している相対アドレス形式の CODE セクション

```
  .SECTION program,CODE,ALIGN=4
```

(3) 絶対アドレス形式の CODE セクション

```
  .SECTION program,CODE
  .ORG 0fff00000H
```

endian=big オプション指定時、本制御命令をセクション属性が CODE のセクションに記述する場合、直前の行にアドレス補正制御命令 (.ALIGN 4) を記述し、4 バイト境界に配置されるようにしてください。記述されていない場合、アセンブラはウォーニングを出力し、自動的に 4 バイト境界に配置します。

.WORD

2 バイト長の固定データを ROM に格納します。

[指定形式]

△ .WORD △ < オペランド >
△ < ラベル名 :> △ .WORD △ < オペランド >

[詳細説明]
2 バイト長の固定データを ROM に格納します。
データを格納したアドレスに、ラベル名を定義することもできます。

例

```
      .SECTION D_2,ROMDATA,ALIGN=2
  symbol:
      .WORD 1
      .WORD symbol
      .WORD symbol+1
      .WORD 1,2,3,4,5
      .END
```

[備考]
本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",ROMDATA" を記述することでセクション属性が ROMDATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドに文字・文字列を記述することはできません。
ラベルを定義する場合には、制御命令の前にラベル名を記述してください。
ラベル名には、必ずコロン (:) を記述してください。

.LWORD

4 バイト長の固定データを **ROM** に格納します。

[指定形式 **]**

△ .LWORD △ < オペランド >
△ < ラベル名 :> △ .LWORD △ < オペランド >

[詳細説明]
4 バイト長の固定データを ROM に格納します。
データを格納したアドレスに、ラベル名を定義することもできます。

例

```
      .SECTION D,ROMDATA,ALIGN=4
  symbol:
      .LWORD 1
      .LWORD symbol
      .LWORD symbol+1
      .LWORD 1,2,3,4,5
      .END
```

[備考]
本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",ROMDATA" を記述することでセクション属性が ROMDATA となります。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
オペランドには数値、シンボル、式を記述できます。
オペランドに文字・文字列を記述することはできません。
ラベルを定義する場合には、制御命令の前にラベル名を記述してください。
ラベル名には、必ずコロン (:) を記述してください。

.FLOAT

4 バイト長の固定データを ROM に格納します。

[指定形式]

△ .FL0AT △ < 数値 >
△ < ラベル名 :> △ .FL0AT △ < 数値 >

[詳細説明]
4 バイト長の固定データを ROM に格納します。
データを格納したアドレスに、ラベル名を定義することもできます。

例
```
      .SECTION D,ROMDATA,ALIGN=4
      .FLOAT 5E2
fcnst: .FLOAT 5e2
      .END
```

[備考]
本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",ROMDATA" を記述することでセクション属性が ROMDATA となります。
オペランドに浮動小数点数を記述してください。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
ラベルを定義する場合には、制御命令の前にラベル名を記述してください。
ラベル名には、必ずコロン (:) を記述してください。

.DOUBLE

8 バイト長の固定データを ROM に格納します。

[指定形式]

△ .DOUBLE △ < 数値 >
△ < ラベル名 :> △ .DOUBLE △ < 数値 >

[詳細説明]
8 バイト長の固定データを ROM に格納します。
データを格納したアドレスに、ラベル名を定義することもできます。

例
```
      .SECTION D,ROMDATA,ALIGN=4
      .DOUBLE 5E2
fdble: .DOUBLE 5e2
      .END
```

[備考]
本制御命令は必ず、ROMDATA 属性のセクション内に記述してください。セクション定義の際に、セクション名に続けて ",ROMDATA" を記述することでセクション属性が ROMDATA となります。
オペランドに浮動小数点数を記述してください。
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
ラベルを定義する場合には、制御命令の前にラベル名を記述してください。

ラベル名には、必ずコロン (:) を記述してください。

.ALIGN

本制御命令を記述した直後の行のコードを格納するアドレスを 2、4 または 8 バイト
アライメントに補正します。

[指定形式]

△ .ALIGN △ < アライメント補正値 >
< アライメント補正値 >:[2|4|8]

[詳細説明]
本制御命令を記述した直後の行のコードを格納するアドレスを 2、4 または 8 バイトアライメントに補正します。
セクション属性が CODE または、ROMDATA の場合は、アドレスを補正した結果、空になったところに NOP のコード（03H）を書き込みます。
セクション属性が DATA の場合は、アドレス補正のみ行います。

例
```
.SECTION PB,CODE,ALIGN=4
MOV.L  R1,R2
.ALIGN 4 ；アドレスを 4 の倍数に補正
RTS
.END
```

[備考]
本制御命令は、次の条件に当てはまるセクション内に記述できます。
(1) セクション定義の際にアドレス補正を指示している相対アドレス形式セクション
```
.SECTION program,CODE,ALIGN=4
```
(2) 絶対アドレス形式セクション
```
.SECTION program,CODE
.ORG 0fff00000H
```
相対アドレス形式のセクションで .SECTION 制御命令行で ALIGN 指定のされていないセクションに本制御命令を記述した場合は、ウォーニングが出力されます。
指定した値がセクションの境界調整数よりも大きい場合は、ウォーニングが出力されます。

5.2.5 マクロ制御擬似命令

制御命令自身はデータを生成しません。命令に対する機械語コードの生成を制御する制御命令です。アドレスの更新は行いません。
マクロ機能および繰り返しマクロ機能を定義するための制御命令です。

表 5.30 マクロ制御命令

制御命令	機能内容
.MACRO	マクロ名を定義します。マクロボディの始まりを定義します。
.EXITM	マクロボディの展開を中止します。
.LOCAL	マクロ内ローカルラベルを宣言します。
.ENDM	マクロボディの終了を示します。
.MREPEAT	繰り返しマクロボディの始まりを示します。
.ENDR	繰り返しマクロボディの終了を示します。
..MACPARA	マクロ呼び出しの実引数の個数を値として持ちます。
..MACREP	繰り返しマクロボディの展開回数を値として持ちます。
.LEN	指定した文字列の文字数を値として持ちます。

制御命令	機能内容
.INSTR	指定した文字列の中で指定した文字列の始まる位置を値として持ちます。
.SUBSTR	指定した文字列の中で指定した位置から指定した文字数分の文字を切り出します。

.MACRO

マクロ名を定義します。

[指定形式]

```
 [ マクロ定義 ]
 △ < マクロ名 > △ .MACRO[< 仮引数 >[,...]]
 △ボディ
 △ .ENDM
 [ マクロ呼び出し ]
 △ < マクロ名 > △ [< 実引数 >[,...]]
```

[詳細説明]
マクロ名を定義します。
マクロ定義の始まりを示します。

例 1
[マクロ定義例]

```
name  .MACRO  string
       .BYTE  'string'
       .ENDM
```

[マクロ呼び出し例 1]

```
   name "name,address"
```

マクロ展開後コード

```
   .BYTE 'name,address'
```

[マクロ呼び出し例 2]

```
   name(name,address)
```

マクロ展開後コード

```
   .BYTE '(name,address)'
```

例 2
[マクロ定義例]

```
   mac .MACRO p1,p2,p3
     .IF ..MACPARA == 3
       .IF 'p1' == 'byte'
         MOV.B #p2,[p3]
       .ELSE
         MOV.W #p2,[p3]
       .ENDIF
     .ELIF ..MACPARA == 2
       .IF 'p1' == 'byte'
         MOV.B #p2,[R3]
       .ELSE
         MOV.W #p2,[R3]
       .ENDIF
     .ELSE
       MOV.W R3,R1
     .ENDIF
   .ENDM
```

[マクロ呼び出し例]
```
  mac word,10,R3
```

マクロ展開後コード
```
  .IF 3 == 3
  .ELSE
    MOV.W #10,[R3]
  .ENDIF
```

[備考]
マクロ名は必ず記述してください。
マクロ名は、「4.1.2　名前」の「名前の記述規則」に従ってください。
マクロ仮引数の名前は、「4.1.2　名前」の「名前の記述規則」に従ってください。
マクロ仮引数の名前は、ネストしているマクロ定義を含めて、異なる名前で定義してください。
仮引数を複数定義する場合は、仮引数をカンマ (,) で区切って記述してください。
制御命令 ".MACRO" のオペランドに記述した仮引数は、必ずマクロボディ内に記述してください。
マクロ名と実引数の間には、必ず空白文字またはタブを記述してください。
実引数は、マクロ呼び出しの際に仮引数に対応させて記述してください。
特殊文字を実引数に記述する場合は、ダブルクォーテーションで囲って記述してください。
実引数には、ラベル、グローバルラベルおよびシンボルが記述できます。
実引数には式が記述できます。
仮引数と実引数は、左から記述されている順に置き換えられます。
仮引数が定義されていて、マクロ呼び出しで実引数の記述がない場合は、仮引数にあたる部分のコードは出力されません。
仮引数の数が、実引数の数より多い場合は、対応する実引数がない仮引数にあたる部分のコードは出力されません。
ボディに記述した仮引数をシングルクォーテーション (') で囲った場合は、対応する実引数をシングルクォーテーションで囲って出力されます。
１つの実引数がカンマ (,) を含む場合に、括弧 (()) で囲った場合は、括弧を含めて変換されます。
実引数の数が、仮引数の数より多い場合は、対応する仮引数がない実引数については処理されません。
ダブルクォーテーションで囲った文字列は、全てその文字列そのものを示します。仮引数をダブルクォーテーションで囲わないでください。
仮引数は 80 個まで記述できます。
1 行に記述できる文字数の範囲内で最大 80 個まで記述できます。
実引数と仮引数の数が合わない場合は、アセンブラはウォーニングメッセージを出力します。

.EXITM

マクロボディの展開を中止し、最も近い ".ENDM" に制御を渡します。

[指定形式]
```
  .EXITM
```

[詳細説明]
マクロボディの展開を中止し、最も近い ".ENDM" に制御を渡します。

例
[マクロ定義例]
```
data1 .MACRO  value
      .IF value == 0
        .EXITM
      .ELSE
        .BLKB value
      .ENDIF
      .ENDM
```

[マクロ呼び出し例]
```
  data1 0
```

マクロ展開後コード
```
  .IF 0 == 0
    .EXITM
```

```
.ENDIF
```

[備考]
マクロ定義のボディ内に記述してください。

.LOCAL

オペランドに記述されたラベルがマクロローカルラベルであることを宣言します。

[指定形式]

```
.LOCAL △ < ラベル名 >[ , . . . ]
```

[詳細説明]
オペランドに記述されたラベルがマクロローカルラベルであることを宣言します。
マクロローカルラベルは、異なるマクロ定義およびマクロ定義外であれば、同一の名前を複数個記述できます。

例
```
name  .MACRO
      .LOCAL  m1; 'm1' is macro local label
m1:
      nop
      bra  m1
      .ENDM
```

[備考]
本制御命令は、必ずマクロボディ内に記述してください。
本制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
本制御命令によるマクロローカルラベル宣言は、ラベル名を定義するより前に記述してください。
マクロローカルラベル名は、「4.1.2　名前」の「名前の記述規則」に従ってください。
本制御命令のオペランドは、カンマで区切って複数のラベルを記述できます。このときの最大ラベル数は 100 個までです。
マクロ定義がネストしている場合は、マクロ定義内で定義を行っているマクロ内のマクロローカルラベルは、同一名を使用できません。
インクルードファイルの内容を含む、1 つのアセンブリ言語ファイルに記述できるマクロローカルラベルは 65535 個までです。

.ENDM

1 つのマクロ定義のボディが終了することを示します。

[指定形式]

```
 < マクロ名 > △ .MACRO
 △ボディ
 △ .ENDM
```

[詳細説明]
1 つのマクロ定義のボディが終了することを示します。

例
[マクロ定義例]
```
lda   .MACRO value
      MOV.L  #value,R3
.ENDM
```

[マクロ呼び出し例]
```
      lda   0    ; MOV.L  #0,R3 に展開される
```

.MREPEAT

繰り返しマクロの始まりを示します。

[指定形式]

```
 [< ラベル >: ] △ .MREPEAT △ < 数値 >
 △ボディ
 △ .ENDR
```

[詳細説明]
繰り返しマクロの始まりを示します。
ボディを指定した数値回、繰り返して展開します。
繰り返し回数は、1 から 65535 の間の数を指定できます。
65535 レベルまでのネストができます。
本制御命令を記述した場所に、マクロボディを展開します。

例
[マクロ定義例]

```
rep   .MACRO  num
      .MREPEAT num
        .IF num > 49
          .EXITM
        .ENDIF
        nop
      .ENDR
      .ENDM
```

[マクロ呼び出し例]

```
 rep  3
```

マクロ展開後コード

```
 nop
 nop
 nop
```

[備考]
オペランドは必ず記述してください。
本制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
本制御命令行の先頭にラベルを記述できます。
オペランドには、シンボルを記述できます。
前方参照となるシンボルは記述できません。
オペランドには、式が記述できます。
ボディには、マクロ定義およびマクロ呼び出しが記述できます。
ボディ内に制御命令 ".EXITM" を記述できます。

.ENDR

繰り返しマクロの終了を示します。

[指定形式]

```
 [< ラベル >: ] △ .MREPEAT △ < 数値 >
 △ボディ
 △ .ENDR
```

[詳細説明]
繰り返しマクロの終了を示します。

[備考]
必ず制御命令 ".MREPEAT" に対応させて記述してください。

..MACPARA

マクロ呼び出しの実引数の個数を示します。

[指定形式]

```
..MACPARA
```

[詳細説明]
マクロ呼び出しの実引数の個数を示します。
".MACRO" によるマクロ定義のボディ内に記述できます。

例
マクロ実引数の数を判断して、条件アセンブルを行います。

```
;[ マクロ定義例 ]
    .GLB mem
name .MACRO  f1,f2
     .IF ..MACPARA == 2
       ADD  f1,f2
     .ELSE
       ADD  f1,R3
     .ENDIF
    .ENDM

;[ マクロ呼び出し例 ]
  name  #mem

; マクロ展開後コード
  .IF 1 == 2
  .ELSE
     ADD  #mem,R3
  .ENDIF
```

[備考]
本制御命令は式の項として記述できます。
".MACRO" によるマクロボディの外に記述した場合、値は 0 となります。

.MACREP

繰り返しマクロが展開されている回数を示します。

[指定形式]

```
.MACREP
```

[詳細説明]
繰り返しマクロが展開されている回数を示します。
".MREPEAT" によるマクロ定義のボディ内に記述できます。
条件アセンブルのオペランドに記述できます。

例
[マクロ定義例]

```
mac   .MACRO  value,reg
      .MREPEAT value
        MOV.B  #0,..MACREP[reg]
      .ENDR
      .ENDM

[ マクロ呼び出し例 ]
  mac 3,R3
```

マクロ展開後コード
```
  MOV.B #0,1[R3]
  MOV.B #0,2[R3]
  MOV.B #0,3[R3]
```

[備考]
本制御命令は式の項として記述できます。
".MACRO" によるマクロボディの外に記述した場合、値は 0 となります。

.LEN

オペランドに記述した文字列の長さを示します。

[指定形式]

```
  .LEN △ {"< 文字列 >"}
  .LEN △ {'< 文字列 >'}
```

[詳細説明]
オペランドに記述した文字列の長さを示します。

例
[マクロ定義例]
```
bufset  .MACRO  f1
buffer: .BLKB  .LEN{'f1'}
        .ENDM
```

[マクロ呼び出し例]
```
  bufset  Sample
```

マクロ展開後コード
```
buffer: .BLKB   6
```

[備考]
オペランドは、必ず {} で囲ってください。
本制御命令とオペランドの間に空白文字またはタブが記述できます。
文字列には、空白文字およびタブを含む文字が記述できます。
文字列は、必ずクォーテーションで囲って記述してください。
本制御命令を式の項に記述できます。
マクロの実引数の文字列長を求める場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲った場合は、仮引数として指定した文字列の長さを示します。

.INSTR

オペランドで指定した文字列のなかで、検出文字列が始まる位置を示します。

[詳細説明]
オペランドで指定した文字列のなかで、検出文字列が始まる位置を示します。
文字列の検索を開始する位置を指定できます。

例
指定した文字列 (japanese) の先頭 (top) からの、"se" 文字列の位置 (7) を取り出します。

[マクロ定義例]
```
top       .EQU   1
point_set .MACRO source,dest,top
point     .EQU .INSTR{'source','dest',top}
          .ENDM
```

[マクロ呼び出し例]
```
  point_set  japanese,se,1
```

マクロ展開後コード
```
point    .EQU   7
```

[備考]
オペランドは、必ず {} で囲ってください。
文字列、検出文字列および検索開始位置は、必ず記述してください。
文字列、検出文字列および検索開始位置は、カンマで区切って記述してください。
カンマの前後には、空白文字およびタブは記述できません。
検索開始位置は、シンボルを記述できます。
検索開始位置を 1 とした場合は、文字列の先頭を示します。
本制御命令は、式の項に記述できます。
文字列よりも、検索文字列が長い場合の値は 0 となります。文字列のなかに、検索文字列が含まれていなかった場合の値は 0 となります。文字列の長さよりも、検索開始位置の値が大きかった場合の値は 0 となります。
マクロの実引数を検出条件としてマクロを展開したい場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲って記述した場合は、その文字列を検出条件としてマクロを展開します。

.SUBSTR

文字列の指定した位置から、指定した文字列を取り出します。

[指定形式]
```
  .SUBSTR △ {  "< 文字列 >",< 切り出し開始位置 >,< 切り出し文字数 >  }
  .SUBSTR △ {  '< 文字列 >',< 切り出し開始位置 >,< 切り出し文字数 >  }
```

[詳細説明]
文字列の指定した位置から、指定した文字列を取り出します。

例
マクロの実引数として与えられた文字列の長さを、".MREPEAT" のオペランドに与えます。
"..MACREP" は、".BYTE" の行を 1 回展開するごとに、1 → 2 → 3 → 4 と増加します。
したがって、マクロの実引数として与えられた文字列の先頭から順に 1 文字ずつ、".BYTE" のオペランドに与えることになります。

[マクロ定義例]
```
name  .MACRO  data
      .MREPEAT .LEN{'data'}
        .BYTE .SUBSTR{'data',..MACREP,1}
      .ENDR
      .ENDM
```

[マクロ呼び出し例]
```
name  ABCD
```

マクロ展開後コード
```
    .BYTE  "A"
    .BYTE  "B"
    .BYTE  "C"
    .BYTE  "D"
```

[備考]
オペランドは、必ず {} で囲ってください。
文字列、切り出し開始位置および切り出し文字数は、必ず記述してください。
文字列、切り出し開始位置および切り出し文字数は、カンマで区切って記述してください。
切り出し開始位置および切り出し文字数には、シンボルが記述できます。切り出し開始位置を 1 とした場合は、文字列の先頭を示します。
文字列には、空白文字およびタブを含む文字が記述できます。
文字列は、必ずクォーテーションで囲って記述してください。

文字列の長さよりも切り出し開始位置の値が大きい場合の値は 0 となります。文字列の長さよりも切り出し文字数の値が大きい場合の値は 0 となります。切り出し文字数を 0 とした場合の値は 0 となります。
マクロの実引数を切り出し条件としてマクロを展開したい場合は、仮引数名をシングルクォーテーションで囲って記述してください。ダブルクォーテーションで囲って記述した場合は、その文字列を切り出し条件としてマクロを展開します。

5.2.6 コンパイラ専用制御擬似命令

コンパイラがアセンブリ言語ソースファイルを生成する際、C 言語の機能をアセンブラで適切に処理させるため、下記の制御命令を出力することがあります。
コンパイラが生成したアセンブリ言語ソースファイルを利用する場合、これらの制御命令の内容を変更せず、そのまま使用してください。また、ユーザアセンブリプログラム作成時には、これらの制御命令は使用しないでください。

表 5.31 コンパイラ専用制御命令

制御命令	内容
._LINE_TOP	#pragma inline_asm で指定された関数を展開した場合に出力されます。
._LINE_END	
.SWSECTION	switch 文で分岐テーブルを使用した場合に出力されます。
.SWMOV	
.SWITCH	
.INSTALIGN	instalign4, instalign8 オプション、または #pragma instalign4, #pragma instalign8 を使用した場合に出力されます。

5.3 制御命令

この節では、制御命令について説明します。
制御命令とは、アセンブラの動作に対し細かい指示を与えるものです。

5.3.1 概要

制御命令は、アセンブラの動作に対し細かい指示を与えるもので、ソース中に記述します。
制御命令は、オブジェクト・コード生成の対象とはなりません。

種類	擬似命令
アセンブルリスト制御命令	.LIST
条件アセンブル制御命令	.IF, .ELIF, .ELSE, .ENDIF
拡張機能制御命令	.ASSERT, ?, @, ..FILE, .STACK, .LINE, .DEFINE

以降、各制御命令について詳細な説明を行います。

5.3.2 アセンブルリスト制御命令

アセンブルリストファイルに出力する情報や、アセンブルリストファイルのフォーマットの制御を行う制御命令です。なお、コード生成には影響を与えません。

表 5.32 アセンブルリスト制御命令

制御命令	機能内容
.LIST	アセンブルリストファイルを生成する際に、アセンブリ言語ファイルの行単位でアセンブルリストファイルへの出力を行うか行わないかを制御します。

.LIST

アセンブルリストファイルへの行の出力を停止 (OFF) することができます。

[指定形式]

```
.LIST △ [ON|OFF]
```

[詳細説明]
アセンブルリストファイルへの行の出力を停止 (OFF) することができます。
リストへの行の出力を停止している範囲においても、エラー発生行についてはアセンブルリストファイルに出力します。
アセンブルリストファイルへの行の出力を開始 (ON) することができます。
本制御命令を指定しない場合は、全ての行をアセンブルリストファイルに出力します。

例
.LIST ON
.LIST OFF

[備考]
制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
行の出力を停止する場合は、オペランドに 'OFF' を記述してください。
行の出力を開始する場合は、オペランドに 'ON' を記述してください。

5.3.3 条件アセンブル制御命令

アセンブル制御擬似命令を使って、指定した範囲の行のアセンブルを行うか、行わないかを指定できます。

.IF, .ELIF, .ELSE, .ENDIF

表 5.33 条件アセンブル制御命令

制御命令	機能内容
.IF	条件アセンブルブロックの始まりを示します。条件の判定を行います。
.ELIF	二つ以上の条件ブロックを記述する場合に、二つ目以降の条件を判定します。
.ELSE	全ての条件が偽である場合に、アセンブルを行うブロックの始まりを示します。
.ENDIF	条件アセンブルブロックの終了を示します。

[指定形式]

```
.IF △条件式
ボディ
.ELIF △条件式
ボディ
.ELSE
ボディ
.ENDIF
```

[詳細説明]
.IF, .ELIF に記述した条件に従いアセンブルを行うブロックを制御します。
.IF, .ELIF のオペランドに記述した条件を判定し、真であれば以降に続くボディをアセンブルします。
条件が真である場合にアセンブルされる行は、制御命令 ".ELIF",".ELSE" および ".ENDIF" 行の前までです。
条件アセンブルブロック内には、アセンブリ言語ファイルに記述可能な全ての命令を記述できます。
条件式の結果によって、条件アセンブルが行われます。

例
[条件式の例]
sym < 1
sym+2 < data1

```
  sym+2 < data1+2
  'smp1' == name
```

[条件アセンブル記述例]

```
  .IF TYPE==0
    .byte "Proto Type Mode"
  .ELIF TYPE>0
    .byte "Mass Produciton Mode"
  .ELSE
    .byte "Debug Mode"
  .ENDIF
```

[備考]
.IF, .ELIF 制御命令のオペランドには、必ず条件式を記述してください。
.IF, .ELIF 制御命令とオペランドの間には、必ず空白文字またはタブを記述してください。
条件式は、制御命令のオペランドに 1 つだけ記述できます。
条件式には、必ず条件演算子を記述してください。
次に示す演算子が記述できます。

表 5.34 .IF および .ELIF 制御命令の条件演算子

条件演算子	内容
＞	左辺値が右辺値より大きい場合に真となります
＜	左辺値が右辺値より小さい場合に真となります
＞＝	左辺値が右辺値より大きいか等しい場合に真となります
＜＝	左辺値が右辺値より小さいか等しい場合に真となります
＝＝	左辺値と右辺値が等しい場合に真となります
！＝	左辺値と右辺値が等しくない場合に真となります

条件式は符号付き 32 ビットで演算します。
条件演算子の左辺および右辺には、シンボルが記述できます。
条件演算子の左辺および右辺には、式が記述できます。式は、「4.1.5 オペランド部の記述方法」の「(2) 式」に従って記述してください。
条件演算子の左辺および右辺には、文字列が記述できます。文字列は、必ずシングルクォーテーション (') またはダブルクォーテーション (") で囲って記述してください。このとき、文字列の大小は、文字コードの値で判定されます。

例)
"ABC"<"CBA" 414243 < 434241 で真となります。
"C" < "A" 43 < 41 で偽となります。

条件演算子の前後には、空白文字またはタブが記述できます。
条件式は、制御命令 ".IF" および ".ELIF" のオペランドに記述できます。
演算結果のオーバフローは判断しません。
シンボルは、前方参照（本制御命令行より後に定義されているシンボルを参照）はできません。
前方参照のシンボルや、未定義のシンボルを記述した場合は、値を 0 として式を判定します。

5.3.4 拡張機能制御命令

コード生成には影響を与えない制御命令です。

表 5.35 拡張機能制御命令

制御命令	機能内容
.ASSERT	オペランドに記述した文字列を標準エラー出力またはファイルに出力します。
?	テンポラリラベルの定義と参照を指定します。
@	@ の前後の文字列を連結し、1 つの文字列として扱います。

制御命令	機能内容
..FILE	アセンブラが処理を行っているアセンブリ言語ファイル名を示します。
.STACK	指定したシンボルに対してスタック値を定義します。
.LINE	行番号を変更します。
.DEFINE	置き換えシンボルを定義します。

.ASSERT

オペランドに記述した文字列をアセンブル時に、標準エラー出力に出力します。
[指定形式]

.ASSERT △ “< 文字列 >”
.ASSERT △ “< 文字列 >”> △ < ファイル名 >
.ASSERT △ “< 文字列 >”>> △ < ファイル名 >

[詳細説明]
オペランドに記述した文字列をアセンブル時に、標準エラー出力に出力します。
ファイル名を指定した場合は、オペランドに記述した文字列をファイルに出力します。
ファイル名に絶対パスを記述した場合は、記述したディレクトリにファイルを生成します。
ファイル名に絶対パスを記述していない場合
(1)output オプションで指定したファイル名にディレクトリ指定がない場合は、本制御命令で指定されたファイルをカレントディレクトリに生成します。
(2)output オプションで指定したファイル名にディレクトリ指定がある場合は、本制御命令で指定されたファイル名に output オプションで指定されたファイルのディレクトリを付加したファイルを生成します。
(3)output オプションが指定されていない場合、アセンブラ起動時にコマンド行で指定したファイルと同じディレクトリにファイルを生成します。
ファイル名に制御命令 ”..FILE” を記述した場合は、アセンブラ起動時にコマンド行で指定したファイルと同じディレクトリにファイルを生成します。

例
sample.dat ファイルにメッセージを出力します。

```
.ASSERT "string" > sample.dat
```

sample.dat ファイルにメッセージを追加します。

```
.ASSERT "string" >> sample.dat
```

現在処理中のファイルと同じ名前で拡張子を除くファイル名のファイルにメッセージを出力します。

```
.ASSERT "string" > ..FILE
```

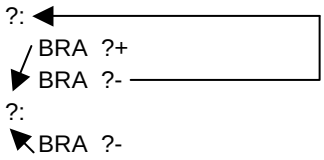
[備考]
オペランドと制御命令の間には、必ず空白文字またはタブを記述してください。
オペランドの文字列は必ずダブルクォーテーションで囲ってください。
文字列をファイルに出力するときは、">" または ">>" に続けてファイル名を指定してください。
> は、新規にファイルを生成して、そのファイルにメッセージを出力します。以前に同一名のファイルがある場合は、そのファイルに上書きされます。
>> は、ファイルの内容に追加して、メッセージを出力します。指定したファイルが存在しない場合は、新しくファイルを生成します。
">" または ">>" の前後には、空白文字またはタブを記述できます。
ファイル名に制御命令 "..FILE" を記述できます。

?

テンポラリラベルを定義します。
[指定形式]

?:
△ < ニーモニック > △ ?+
△ < ニーモニック > △ ?-

[詳細説明]
テンポラリラベルを定義します。
直前または直後に定義されたテンポラリラベルを参照します。
同一ファイル内で定義および参照が可能です。
ファイル内に 65535 個までのテンポラリラベルが定義できます。このとき、ファイル内に ".INCLUDE" が記述されている場合は、インクルードファイル内のテンポラリラベルを含み 65535 個までの記述ができます。
アセンブルリストファイルには、テンポラリラベルとして変換された結果が出力されます。
例



矢印のテンポラリラベルを指す

[備考]
テンポラリラベルとして定義したい行に "?:" を記述してください。
直前に定義したテンポラリラベルを参照したい場合は、命令のオペランドに "?-" を記述してください。
直後に定義したテンポラリラベルを参照したい場合は、命令のオペランドに "?+" を記述してください。
参照できるラベルは、直前と直後のラベルだけです。

@

マクロ引数、マクロ変数、予約シンボル、制御命令 "..FILE" の展開ファイル名および指定文字列を連結します。

[指定形式]

< 文字列 >@< 文字列 >[@< 文字列 > . . .]

[詳細説明]
マクロ引数、マクロ変数、予約シンボル、制御命令 "..FILE" の展開ファイル名および指定文字列を連結します。

例
ファイル名の連結例 :
現在処理中のファイル名が sample1.src の場合、sample.dat ファイルにメッセージを出力します。

```
 .ASSERT   "sample" > ..FILE@.dat
```

文字列の連結例 :

```
mov_nibble .MACRO p1,src,dest  ; マクロ定義
          MOV.@p1  src,dest
          .ENDM

    mov_nibble  W,R1,R2       ; マクロ呼び出し

    MOV.W R1,R2               ; マクロ展開後コード
```

[備考]
本制御命令の前後に記述した空白文字およびタブは、文字列として連結します。
本制御命令の前後には、文字列が記述できます。
@ を文字データ (40H) として記述する場合は、ダブルクォーテーション (") で囲んでください。@ を含む文字列をシングルクォーテーション (') で囲った場合は、@ の前後の文字列を連結します。
一行に複数回記述できます。
連結した文字列を名前とする場合は、本制御命令の前後に空白文字およびタブを記述しないでください。

..FILE

アセンブラが処理中のファイル名に展開されます（アセンブリ言語ファイル名またはインクルードファイル）。

[指定形式]

```
..FILE
```

[詳細説明]
アセンブラが処理中のファイル名に展開されます（アセンブリ言語ファイル名またはインクルードファイル）。

例
アセンブリ言語ファイル名が "sample.src" の場合、"sample" ファイルにメッセージを出力します。
.ASSERT "sample" > ..FILE

アセンブリ言語ファイル名が "sample.src" の場合、"sample.inc" ファイルをインクルードします。
.INCLUDE ..FILE@.inc

上記の行が、"sample.src" ファイルでインクルードしている "incl.inc" 内に記述されている場合、通常、"incl.mes" に文字列を出力します。
.ASSERT "sample" >..FILE@.mes

[備考]
制御命令 ".ASSERT" および制御命令 ".INCLUDE" のオペランドに記述できます。
本制御命令で読み込まれるファイル名は、ファイルの拡張子およびパスを除いた部分です。

.STACK

シンボルに対して、Call Walker で表示するスタック使用量を定義します。

[指定形式]

```
.STACK △ < 名前 >=< 数値 >
```

[詳細説明]
シンボルに対して、Call Walker で表示するスタック使用量を定義します。

例
.STACK SYMBOL=100H

[備考]
1 つのシンボルに対して定義できるスタック値は 1 度のみ有効とします。
2 度以上指定した場合は、その定義を無効とします。また、指定できるスタック値は、
0H ～ 0FFFFFFFCH の範囲の 4 の倍数のみとし、それ以外を指定した場合はその定義を無効とします。
＜数値＞は定数値とし、かつ前方参照シンボル、外部参照シンボル、相対アドレスシンボルを使わずに指定してください。

.LINE

アセンブラのエラーメッセージあるいは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。

[指定形式]

```
.LINE △ < ファイル名 >,< 行番号 >
.LINE △ < 行番号 >
```

[詳細説明]
アセンブラのエラーメッセージあるいは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。
プログラム内の最初の .LINE 以降は次の .LINE まで行番号、ファイル名を更新しません。
コンパイラは、デバッグオプションを指定してアセンブリ言語ファイルを出力する時に C 言語ソースファイル行に対応する .LINE を生成します。
ファイル名を省略するとファイル名は変更されず、行番号だけが変更されます。

例
.LINE "C:\asm\test.c",5

.DEFINE

シンボルに文字列を定義します。

[指定形式]

```
< シンボル名 > △ .DEFINE △ < 文字列 >
< シンボル名 > △ .DEFINE △ '< 文字列 >'
< シンボル名 > △ .DEFINE △ "< 文字列 >"
```

[詳細説明]
シンボルに文字列を定義します。
シンボルは再定義が可能です。

例
```
X_HI  .DEFINE   R1
    MOV.L  #0,X_HI
```

[備考]
空白文字またはタブを含む文字列を定義する場合は、必ずシングルクォーテーション (') または、ダブルクォーテーション (") で囲って記述してください。
本制御命令で定義されたシンボルは、外部参照指定ができません。
本制御命令と .EQU 制御命令で同名のシンボルを宣言した場合、先に記述した方が優先されます。

5.4 マクロ名

オプション指定やバージョンに合わせて、以下のようなプリデファインドマクロが定義されます。

表 5.36 アセンブラのプリデファインドマクロ

	マクロ名	値	オプション
1	__RX600 __RX200	.DEFINE 1 .DEFINE 1	cpu=rx600 cpu=rx200
2	__BIG __LITTLE	.DEFINE 1 .DEFINE 1	endian=big endian=little
3	__RENESAS_VERSION__ *1	.DEFINE *XXYYZZ*00H *2	-
4	__RXV1	.DEFINE 1	isa=rxv1 *3
5	__RXV2	.DEFINE 1	isa=rxv2 *3
6	__RXV3 【V3.00.00 以降】	.DEFINE 1	isa=rxv3 *3
7	__RX_ISA_VERSION__ 【V3.00.00 以降】	.DEFINE 1 .DEFINE 2 .DEFINE 3	isa=rxv1 *3 isa=rxv2 *3 isa=rxv3 *3
8	__ASRX__ *1【V2.03.00 以降】	.DEFINE 1	-
9	__RENESAS__ *1【V2.03.00 以降】	.DEFINE 1	-
10	__FPU	.DEFINE 1	-fpu
11	__DPFPU	.DEFINE 1	-dpfpu

注 1. オプションにかかわらず常に定義されます。

注 2. アセンブラのバージョンが V*XX.YY.ZZ* の場合、__RENESAS_VERSION__ の値は *XXYYZZ*00H となります。
例) V3.01.00 の場合、__RENESAS_VERSION__ .DEFINE 03010000H

注 3. 環境変数 ISA_RX による指定を含みます。

5.5 予約語

アセンブラでは、アセンブラ制御命令やニーモニックなどと同一の文字列を予約語として扱います。予約語は特別な機能を持っているため、アセンブリ言語ファイル中でラベル名やシンボル名に使用することはできません。また、予約語は大文字と小文字を区別しません。"ABS" と "abs" は同じ予約語となります。

予約語には以下のものがあります。

(1) アセンブラ制御命令
アセンブラ制御命令と、ピリオド (.) で始まる文字列全てを予約語とします。

(2) ニーモニック
RX ファミリのニーモニック全てを予約語とします。

(3) レジスタ・フラグ名
RX ファミリのレジスタ名およびフラグ名全てを予約語とします。

(4) 演算子
本章で説明している全ての演算子を予約語とします。

(5) システムラベル
アセンブラが生成する、ピリオド 2 つから始まる名前をシステムラベルといい、システムラベルは全て予約語として扱います。

6. セクション仕様

6.1 セクション名一覧

CC-RX で扱うセクションについて説明します。
アセンブラが出力するリロケータブルファイルの実行命令、データの各領域は、セクションを構成します。セクションは、メモリ上の配置を行う最小単位です。セクションの性質には、以下の項目があります。

- セクション属性
  code　　実行命令を格納します。
  data　　変更可能なデータを格納します。
  romdata　固定データを格納します。
- 形式種別

  相対アドレス形式　・・・・　最適化リンケージエディタで再配置可能なセクションです。

  絶対アドレス形式　・・・・　アドレス決定済みのセクションです。最適化リンケージエディタで再配置できません。

- 初期値
  プログラム実行開始時の初期値の有無です。同一セクション内に、初期値があるデータと初期値がないデータを混在させることはできません。例えば、ある配列の初期化リストが、その要素数未満である場合、初期化リストで初期値を指定されなかった要素は、初期値として 0 が指定された場合と同様に扱います。
- 書き込み操作
  プログラム実行時における書き込み操作の可 / 不可を示します。
- アライメント数
  セクションの配置アドレスを補正するための値です。最適化リンケージエディタでは、各セクションの配置アドレスを、それぞれのアライメント数の倍数になるように補正します。

6.1.1 C/C++ プログラムのセクション

C/C++ プログラム、標準ライブラリの使用メモリ領域の種類とセクションとの対応を表 6.1 に示します。

表 6.1　メモリ領域の種類とその性質の概要

No.	名称	セクション		形式種別	初期値 書き込み操作	アライメント数	内容
		名称	属性				
1	プログラム領域	P *1 *6	code	相対	有 不可	1byte *7	機械語を格納
2	定数領域	C_8 *1 *2 *6 *8 *10	romdata	相対	有 不可	8byte	const 型のデータを格納
		C *1 *2 *6 *8	romdata	相対	有 不可	4byte	
		C_2 *1 *2 *6 *8	romdata	相対	有 不可	2byte	
		C_1 *1 *2 *6 *8	romdata	相対	有 不可	1byte	

No.	名称	セクション		形式種別	初期値 書き込み操作	アライメント数	内容
		名称	属性				
3	初期化データ領域	D_8 *1 *2 *6 *8 *10	romdata	相対	有 可	8byte	初期値のあるデータを格納
		D *1 *2 *6 *8	romdata	相対	有 可	4byte	
		D_2 *1 *2 *6 *8	romdata	相対	有 可	2byte	
		D_1 *1 *2 *6 *8	romdata	相対	有 可	1byte	
4	未初期化データ領域	B_8 *1 *2 *6 *8 *10	data	相対	無 可	8byte	初期値のないデータを格納
		B *1 *2 *6 *8	data	相対	無 可	4byte	
		B_2 *1 *2 *6 *8	data	相対	無 可	2byte	
		B_1 *1 *2 *6 *8	data	相対	無 可	1byte	
5	switch 文分岐テーブル領域	W *1 *2	romdata	相対	有 不可	4byte	switch 文の分岐テーブルを格納
		W_2 *1 *2	romdata	相対	有 不可	2byte	
		W_1 *1 *2	romdata	相対	有 不可	1byte	
6	C++ 初期処理／後処理データ領域	C$INIT	romdata	相対	有 不可	4byte	グローバルクラスオブジェクトに対して呼び出されるコンストラクタおよびデストラクタのアドレスを格納
7	C++ 仮想関数表領域	C$VTBL	romdata	相対	有 不可	4byte	クラス宣言中に仮想関数があるときに仮想関数をコールするためのデータを格納
8	ユーザスタック領域	SU	data	相対	無 可	4byte	プログラム実行に必要な領域
9	割り込みスタック領域	SI	data	相対	無 可	4byte	プログラム実行に必要な領域
10	ヒープ領域	—	—	相対	無 可	—	ライブラリ関数 malloc、realloc、calloc、new で使用する領域 *9
11	絶対アドレス変数領域	$ADDR_ <section>_ <address> *3	data	絶対	有 / 無 可 / 不可 *4	—	#pragma address 指定した変数を格納
12	可変ベクタ領域	C$VECT C$VECT< ベクタテーブル番号 >	romdata	相対	無 可	4byte	可変ベクタテーブル

No.	名称	セクション		形式種別	初期値 書き込み操作	アライメント数	内容
		名称	属性				
13	リテラル領域	L *5	romdata	相対	有 可 / 不可	4byte	文字列リテラルおよび集成体の動的初期化で用いる初期化子を格納

注 1. **section** オプションでセクション名を切り替えることができます。

注 2. セクション名切り替えの際に、アライメント数が 4 のセクションを指定することで、アライメントが 1、2 または 8 のセクション名も変更されます。

注 3. <section> は C,D,B のセクション名称、<address> は絶対アドレス値 (16 進数) になります。

注 4. 初期値、書き込み操作は <section> の属性に従います。

注 5. section オプションでセクション名を変更することができます。このとき、変更後の名前に C セクションを選択することも可能です。

注 6. #pragma section でセクション名を変更することができます。

注 7. instalign4 オプション、instalign8 オプション、#pragma instalign4 または #pragma instalign8 のいずれかを使用すると、アライメント数は 4 または 8 になります。

注 8. #pragma endian で endian オプションと異なる指定のエンディアンを指定した場合、#pragma endian big であれば _B を、#pragma endian little であれば _L を、セクション名の後ろに付加した専用のセクションを生成し、該当データを格納します。

注 9. これらの関数を使用するためには、最小で 16 バイトのヒープ領域が必要です。

注 10. dpfpu オプション指定時に倍精度浮動小数点型データを格納するためのセクションです。

例 1. C プログラムとコンパイラ生成セクションとの対応をプログラム例を用いて示します。

C プログラム

```
int a=1;
char b;
const short c=0;
void main(){
 ...
}
```

コンパイラが生成する領域と格納されるデータ	セクション名
プラグラム領域 (main(){...})	P
定数領域 (c)	C_2
初期化データ領域 (a)	D
未初期化データ領域 (b)	B_1

例 2. C++ プログラムとコンパイラ生成セクションとの対応をプログラム例を用いて示します。

C++ プログラム

```
class A{
  int m;
  A(int p);
  ~A();
};
A a(1);
char b;
extern const char c='a';
short d=1;
void f(){...}
```

コンパイラが生成する領域と格納されるデータ	セクション名
プログラム領域 (f(){...})	P
定数領域 (c)	C_1
初期化データ領域 (d)	D_2
未初期化データ領域 (a) 未初期化データ領域 (b)	B B_1
初期処理／後処理データ領域 (&A::A, &A::~A)	C $INT

6.2 アセンブリプログラムのセクション

アセンブリプログラムでは、.SECTION 制御命令を用いてセクションの開始や属性を、.ORG 制御命令を用いてセクションの形式種別を、それぞれ宣言します。
各制御命令の詳細については「5.2 擬似命令」を参照してください。

例　　アセンブリプログラムのセクション宣言例を示します。

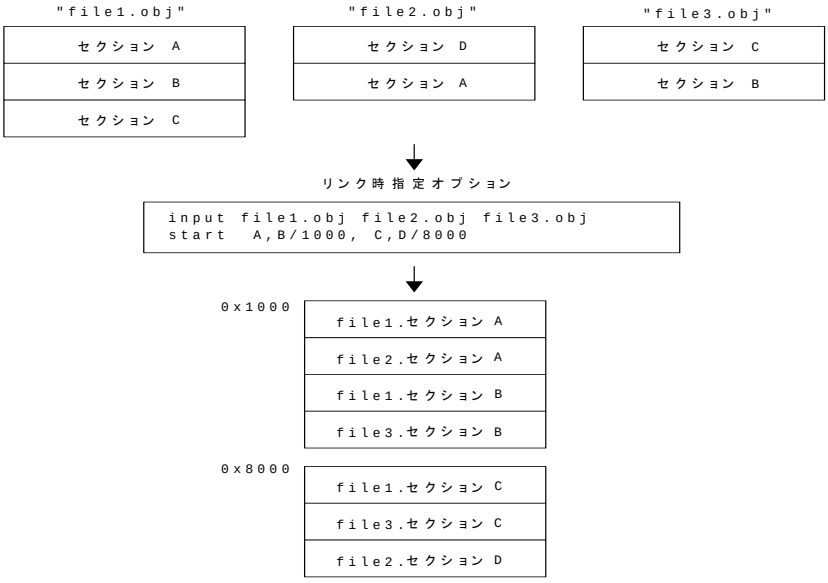
```
                .SECTION        A,CODE,ALIGN=4      ;(1)

START:
                MOV.L           #CONST,R4
                MOV.L           [R4],R5
                ADD             #10,R5,R3
                MOV.L           #100,R4
                MOV.L           #ARRAY,R5
LOOP:
                MOV.L           R3,[R5+]
                SUB             #1,R4
                CMP             #0,R4
                BNE             LOOP
EXIT:
                RTS

;
                .SECTION        B,ROMDATA           ;(2)
                .ORG            02000H
                .glb            CONST
CONST:
                .LWORD          05H
;
                .SECTION        C,DATA,ALIGN=4      ;(3)
                .glb            BASE
BASE:
                .blkl           100
                .END
```

(1) セクション名 A、アライメント数 4、相対アドレス形式の code セクションを宣言しています。
(2) セクション名 B、割り付けアドレス 2000H、絶対アドレス形式の romdata セクションを宣言しています。
(3) セクション名 C、アライメント数 4、相対アドレス形式の data セクションを宣言しています。

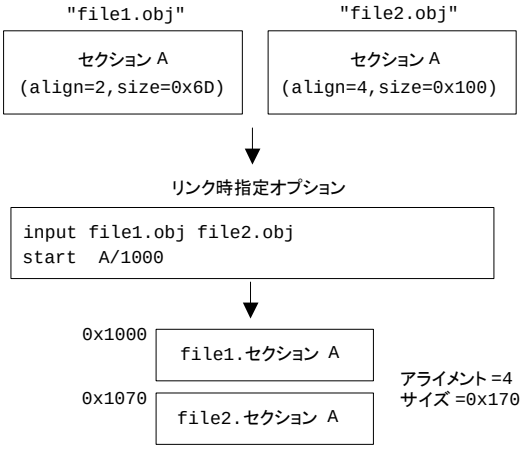
6.3 セクションの結合

最適化リンケージエディタでは、入力リロケータブルファイル内の同一セクションを結合し、start オプションによって指定されたアドレスに割り付けます。

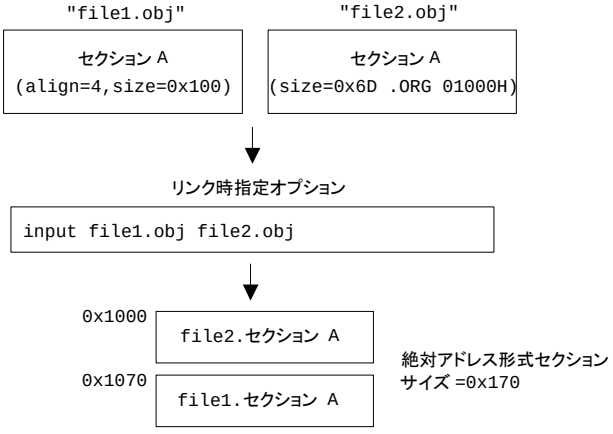
[1] 異なるファイルの同名セクションは、ファイルの入力順に連続して割り付けます。



[2] アライメント数の異なる同名セクションは、アライメント調整後に結合します。セクションのアライメント数は大きい方に合わせます。

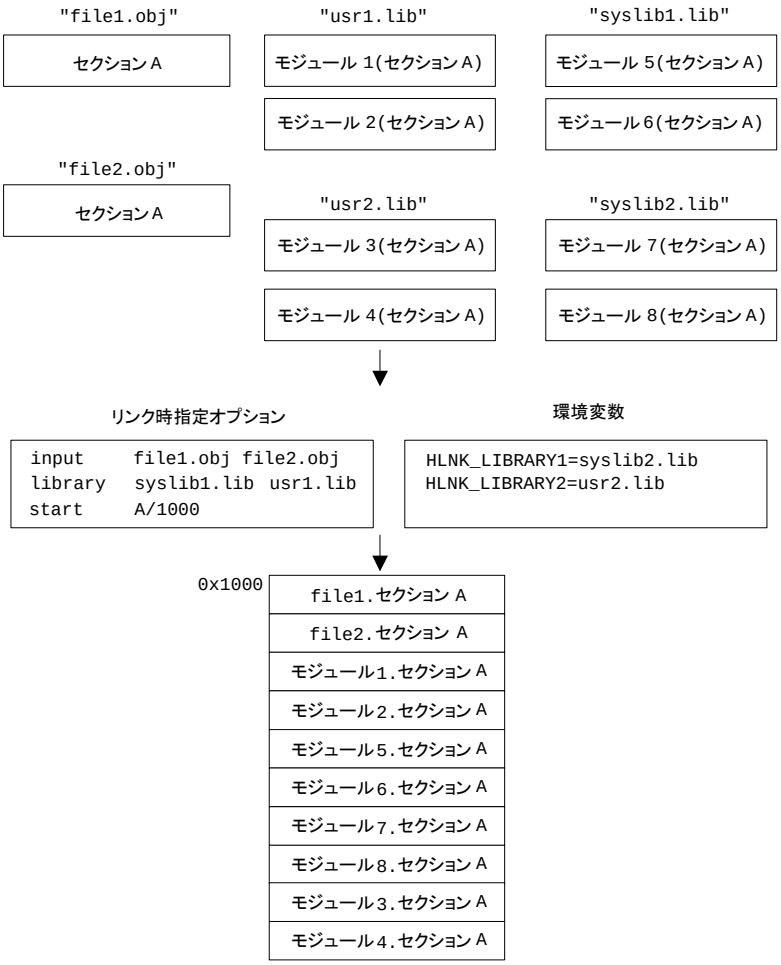


[3] 同名セクションに絶対アドレス形式と相対アドレス形式が含まれている場合、絶対アドレス形式セクションの後に相対アドレス形式セクションを結合します。



[4] 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通りです。

- 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
- 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
- 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_1 ～ 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



7. ライブラリ関数仕様

この章では、CC-RX 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-RX で提供しているライブラリは、C 標準ライブラリ、C99 標準ライブラリ、および EC++ ライブラリです。

7.1.1 ライブラリ関数の説明で使用する用語

(1) ストリーム入出力
データの入出力において、1 文字ごとの入出力関数の呼び出しのたびに入出力装置を駆動したり、OS の機能を呼び出していたのでは、効率が悪くなります。そこで、通常はバッファと呼ばれる記憶域を用意しておき、バッファ内のデータに対して一括して入出力を行います。
一方、プログラムの側から見ると、1 文字ごとに入出力関数を呼び出せた方が便利です。
ライブラリ関数では、バッファの管理を自動的に行うことにより、プログラム内でバッファの状態を意識することなしに、1 文字単位の入出力を効率よく行うことができます。
このように、データの入出力を効率よく実現するために詳細な手段を意識せず、入出力をひとつのデータの流れ (ストリーム) と考えてプログラムを作ることのできる機能をストリーム入出力といいます。

(2) FILE 構造体およびファイルポインタ
ストリーム入出力に必要なバッファやその他の情報は、一つの構造体の中に記憶されており、標準インクルードファイル <stdio.h> の中で FILE という名前で定義されています。
ストリーム入出力においては、ファイルはすべて FILE 構造体のデータ構造を持つものとして扱います。このようなファイルをストリームファイルと呼びます。このファイル構造体へのポインタをファイルポインタと呼び、入出力ファイルを指定するために用います。
ファイルポインタは、

```
FILE *fp;
```

と定義します。
fopen 関数等でファイルをオープンすると、ファイルポインタが得られますが、オープン処理が失敗すると NULL が返ってきます。NULL ポインタを、他のストリーム入出力関数に指定すると、その関数は異常終了しますので、注意が必要です。ファイルをオープンした時は、必ずファイルポインタの値をチェックするようにしてください。

(3) 関数とマクロ
ライブラリ関数の実現方法としては、関数とマクロの二通りがあります。
関数は、通常のユーザ作成の関数と同じインタフェースを持ち、リンク時に取り込みます。
マクロは、その関数に関連した標準インクルードファイルの中で #define 文を用いて定義されています。
マクロについては、以下の点に注意する必要があります。

- マクロは、プリプロセッサによって自動的に展開されてしまうので、ユーザが同じ名前の関数を宣言してもマクロを無効にすることはできません。
- マクロのパラメータとして副作用のある式 (代入式、インクリメント、デクリメント) を指定した時、その効果は保証しません。

例：
パラメータの絶対値を求める MACRO を以下のようにマクロ定義します。

```
#define MACRO(a) ((a)>=0?(a):-(a))
```

と定義されている時、

```
X=MACRO(a++)
```

がプログラム内にあると、

```
X=((a++)>=0?(a++):-(a++))
```

と展開され、a は 2 回インクリメントされることになり、また結果の値も最初の a の値の絶対値とは異なります。

(4) EOF
getc 関数、getchar 関数、fgetc 関数等のファイルからデータを入力する関数において、ファイル終了 (End Of File) 時に返される値です。EOF は、標準インクルードファイル <stdio.h> の中で定義されています。

(5) NULL
ポインタが何も指していない時の値です。NULL は、標準インクルードファイル <stddef.h> の中で定義されています。

(6) ヌル文字
C/C++ 言語における文字列の終わりは、文字 "\0" によって示されることになっています。
ライブラリ関数における文字列のパラメータも、すべてこの約束に従っていなければなりません。
この文字列の終わりを示す文字 "\0" を以下ヌル文字と呼びます。

(7) リターンコード
ライブラリ関数の中には、リターン値によって、指定された処理が成功したか、失敗したか等の結果を判断するものがあります。
このような場合のリターン値を特にリターンコードと呼びます。

(8) テキストファイルとバイナリファイル
多くのシステムでは、データを格納するために特殊なファイル形式を持っています。
これをサポートするために、ライブラリ関数にはテキストファイルとバイナリファイルの 2 種類のファイル形式があります。

- テキストファイル
  テキストファイルは、通常のテキストを格納するためのファイルで、行の集まりとして構成されています。テキストファイルの入力の時、行の区切りとして改行文字 ("\n") が入力されます。また、出力の時、改行文字を出力することにより、現在の行の出力を終了します。テキストファイルは、処理系ごとの標準的なテキストを格納するファイルの入出力を行うためのファイルです。テキストファイルでは、ライブラリ関数で入出力する文字は必ずしもファイル内の物理的なデータの並びと対応していません。

- バイナリファイル
  バイナリファイルは、バイトデータの列として構成されているファイルです。ライブラリ関数で入出力するデータは、ファイル内の物理的なデータの並びと対応しています。

(9) 標準入出力ファイル
入出力のライブラリ関数で、ファイルのオープン等の準備を行わずに標準的に使用できるファイルを標準入出力ファイルといいます。標準入出力ファイルには、標準入力ファイル (stdin)、標準出力ファイル (stdout)、標準エラー出力ファイル (stderr) があります。

- 標準入力ファイル (stdin)
  プログラムへの入力となる標準的なファイルです。

- 標準出力ファイル (stdout)
  プログラムからの出力となる標準的なファイルです。

- 標準エラー出力ファイル (stderr)
  プログラムからのエラーメッセージ等の出力を行うための標準的なファイルです。

(10) 浮動小数点型
浮動小数点型は、実数を近似して表現したものです。C/C++ 言語のソースプログラム上では浮動小数点型を 10 進数で表現していますが、計算機の内部では通常 2 進数で表現されます。
2 進数の場合の浮動小数点型の表現は次のようになります。
$2^n \times m$ (n：整数、m：2 進小数)
ここで n を浮動小数点型の指数部、m を仮数部といいます。浮動小数点型を一定のデータサイズで表現するために、n と m のビット数は通常固定されています。
以下、浮動小数点型に関する用語を説明します。

- 基数
  浮動小数点型が何進数で表現されているかを示す整数値です。通常、基数は 2 です。

- 丸め
  浮動小数点型よりも精度の高い演算の途中結果を浮動小数点型に格納する場合に丸めが行われます。丸めには、切り上げ、切り捨て、四捨五入 (2 進小数の場合は、0 捨 1 入となります) があります。

- 正規化
  浮動小数点型を、$2^n \times m$ の形式で表現する場合、同一の数値を表す異なる表現が可能です。
  例：
  $2^5 \times 1.0_{(2)}$ ($_{(2)}$ は 2 進数を示します)
  $2^6 \times 0.1_{(2)}$
  どちらも同じ値です。
  通常は、有効桁数を確保するために、先頭の桁が 0 でないような表現を用います。これを正規化された浮動小数点型といい、浮動小数点型をこのような表現に変換する操作を正規化といいます。

- ガードビット
  浮動小数点型の演算の途中結果を保持する場合、通常は、丸めを行うために実際の浮動小数点型よりも 1 ビット多いデータを用意します。しかし、これだけでは桁落ち等が生じた時に正確な結果を求めることができません。このために、もう 1 ビット設けて演算の途中結果を保持する手法があります。このビットをガードビットといいます。

(11) ファイルアクセスモード
ファイルをオープンする時にどのような処理をファイルに行うかを示す文字列です。文字列の種類には表 7.1 に示す 12 種類があります。

表 7.1 ファイルアクセスモードの種類

	アクセスモード	意味
1	“r”	テキストファイルを読み込み用にオープンします。
2	“w”	テキストファイルを書き出し用にオープンします。
3	“a”	テキストファイルを追加用にオープンします。
4	“rb”	バイナリファイルを読み込み用にオープンします。
5	“wb”	バイナリファイルを書き出し用にオープンします。
6	“ab”	バイナリファイルを追加用にオープンします。
7	“r+”	テキストファイルを読み込み用でかつ更新用にオープンします。
8	“w+”	テキストファイルを書き出し用でかつ更新用にオープンします。
9	“a+”	テキストファイルを追加用でかつ更新用にオープンします。
10	“r+b”	バイナリファイルを読み込み用でかつ更新用にオープンします。
11	“w+b”	バイナリファイルを書き出し用でかつ更新用にオープンします。
12	“a+b”	バイナリファイルを追加用でかつ更新用にオープンします。

(12) 処理系定義
コンパイラが異なることによって定義が異なるという意味です。

(13) エラー指示子、ファイル終了指示子
ストリームファイルごとに、ファイルの入出力の際にエラーが生じたかどうかを示すエラー指示子と、入力ファイルが終了したかどうかを示すファイル終了指示子というデータを保持しています。
これらのデータは、それぞれ ferror 関数、feof 関数によって参照することができます。
ストリームファイルを扱う関数のうち、そのリターン値だけからではエラーの発生やファイルの終了の情報が得られないものがあります。エラー指示子とファイル終了指示子は、このような関数の実行後にファイルの状態を調べるために使用することができます。

(14) 位置指示子
ディスク上のファイル等、ファイル内の任意の位置からの読み書きができるストリームファイルは、現在読み書きしているファイル内の位置を示すデータを保持しています。これを位置指示子といいます。端末装置等、ファイル内の読み書きの位置を変更できないストリームファイルでは、位置指示子は使用しません。

7.1.2 ライブラリ使用時の注意事項

ライブラリの中で定義されているマクロの内容は、コンパイラごとに異なります。
ライブラリを使用する場合、これらのマクロの内容を再定義した場合、動作は保証しません。
ライブラリは、すべての場合についてエラーを検出しているわけではありません。以降の説明に示す以外の形式でライブラリ関数を呼び出した場合、動作は保証しません。

7.2 ヘッダ・ファイル

RX でライブラリを使用するときに必要となるヘッダ・ファイルの一覧は次のとおりです。なお、各ファイルにはマクロ定義、関数宣言が記述されています。

表 7.2 ライブラリの種類と対応する標準インクルードファイル

	ライブラリの種類	内容	標準インクルードファイル
1	プログラム診断用ライブラリ	プログラムの診断情報の出力を行うライブラリです。	<assert.h>
2	文字操作用ライブラリ	文字の操作およびチェックを行うライブラリです。	<ctype.h>
3	数値計算用ライブラリ	三角関数等の数値計算を行うライブラリです。	<math.h> <mathf.h>
4	プログラムの制御移動用ライブラリ	関数間の制御の移動をサポートするライブラリです。	<setjmp.h>
5	可変個の実引数アクセス用ライブラリ	可変個の実引数を持つ関数に対し、その実引数へのアクセスをサポートするライブラリです。	<stdarg.h>
6	入出力用ライブラリ	入出力操作を行うライブラリです。	<stdio.h>
7	標準処理用ライブラリ	記憶域管理等の C プログラムでの標準的処理を行うライブラリです。	<stdlib.h>
8	文字列操作用ライブラリ	文字列の比較、複写等を行うライブラリです。	<string.h>
9	複素数計算ライブラリ	複素数の計算を行うライブラリです。	<complex.h>
10	浮動小数点環境ライブラリ	浮動小数点環境のライブラリです。	<fenv.h>
11	整数型の書式変換	最大幅の整数の操作、変換を行うライブラリです。	<inttypes.h>
12	多バイト文字、ワイド文字ライブラリ	多バイト文字の操作を行うライブラリです。	<wchar.h> <wctype.h>

また、以上の標準インクルードファイルの他にプログラムの作成作業の効率向上を図るため表 7.3 に示すマクロ名の定義だけからなる標準インクルードファイルがあります。

表 7.3 マクロ名定義からなる標準インクルードファイル

	標準インクルードファイル	内容
1	<stddef.h>	各標準インクルードファイルで共通に使用するマクロ名を定義します。
2	<limits.h>	コンパイラの内部処理に関する各種制御値を定義します。
3	<errno.h>	ライブラリ関数においてエラーが発生した時に errno に設定する値を定義します。
4	<float.h>	浮動小数点型の限界に関する各種制御値を定義します。
5	<iso646.h>	代替つづりのマクロ名を定義します。
6	<stdbool.h>	論理型、および論理値に関するマクロを定義します。
7	<stdint.h>	指定した幅の整数型を宣言してマクロを定義します。
8	<tgmath.h>	型総称マクロを定義します。

7.3 リエントラント性

標準ライブラリ構築ツールで -reent オプションを指定して作成したライブラリは、rand、srand 関数および EC++ ライブラリを除いてすべてリエントラントに実行できます。

-reent オプションを指定していない場合について、表 7.4 および表 7.5 にリエントラントライブラリ一覧を示します。表中、△で示した関数は、errno 変数を設定しますので、プログラム中で errno を参照していなければリエントラントに実行できます。

リエントラント欄 ○：リエントラント ×：ノンリエントラント △：errno を設定

表 7.4 C(C89) リエントラントライブラリ関数一覧

標準インクルードファイル	関数名	リエントラント	標準インクルードファイル	関数名	リエントラント
stddef.h	offsetof	○	math.h	frexp	△
assert.h	assert	×		ldexp	△
ctype.h	isalnum	○		log	△
	isalpha	○		log10	△
	iscntrl	○		modf	△
	isdigit	○		pow	△
	isgraph	○		sqrt	△ *1
	islower	○		ceil	△
	isprint	○		fabs	△ *1
	ispunct	○		floor	△
	isspace	○		fmod	△
	isupper	○	mathf.h	acosf	△ *2
	isxdigit	○		asinf	△ *2
	tolower	○		atanf	△ *2
	toupper	○		atan2f	△ *2
math.h	acos	△ *2		cosf	△ *2
	asin	△ *2		sinf	△ *2
	atan	△ *2		tanf	△ *2
	atan2	△ *2		coshf	△
	cos	△ *2		sinhf	△
	sin	△ *2		tanhf	△
	tan	△ *2		expf	△
	cosh	△		frexpf	△
	sinh	△		ldexpf	△
	tanh	△		logf	△
	exp	△		log10f	△

標準インクルードファイル	関数名	リエントラント
mathf.h	modff	△
	powf	△
	sqrtf	△ *1
	ceilf	△
	fabsf	○
	fabsl	△ *1
	floorf	△
	fmodf	△
setjmp.h	setjmp	○
	longjmp	○
stdarg.h	va_start	○
	va_arg	○
	va_end	○
stdio.h	fclose	×
	fflush	×
	fopen	×
	freopen	×
	setbuf	×
	setvbuf	×
	fprintf	×
	fscanf	×
	printf	×
	scanf	×
	sprintf	×
	sscanf	△
	vfprintf	×
	vprintf	×
	vsprintf	×
	fgetc	×
	fgets	×
	fputc	×
	fputs	×
	getc	×
	getchar	×

標準インクルードファイル	関数名	リエントラント
stdio.h	gets	×
	putc	×
	putchar	×
	puts	×
	ungetc	×
	fread	×
	fwrite	×
	fseek	×
	ftell	×
	rewind	×
	clearerr	×
	feof	×
	ferror	×
	perror	×
stdlib.h	atof	△
	atoi	△
	atol	△
	atoll	△
	strtod	△
	strtol	△
	strtoul	△
	strtoll	△
	strtoull	△
	rand	×
	srand	×
	calloc	×
	free	×
	malloc	×
	realloc	×
	bsearch	○
	qsort	○
	abs	○
	div	○
	labs	○

標準インクルードファイル	関数名	リエントラント	標準インクルードファイル	関数名	リエントラント
stdlib.h	llabs	○	string.h	strchr	○
	ldiv	○		strcspn	○
	lldiv	○		strpbrk	○
string.h	memcpy	○		strrchr	○
	strcpy	○		strspn	○
	strncpy	○		strstr	○
	strcat	○		strtok	×
	strncat	○		memset	○
	memcmp	○		strerror	○
	strcmp	○		strlen	○
	strncmp	○		memmove	○
	memchr	○			

注 1. 関数呼び出しが命令に置き換わる場合は errno 変数を更新しないため ○ になります。
命令に置き換わる条件は、コンパイルオプションの -library の項目を参照ください。

注 2. 三角関数演算器を利用するコードに置き換わると ×(ノンリエントラント) になる場合があります。詳細は、コンパイルオプションの -tfu の項目を参照ください。

表 7.5 C99 リエントラントライブラリ関数一覧

標準インクルードファイル	関数名	リエントラント		標準インクルードファイル	関数名	リエントラント
stddef.h	isblank	○		math.h	sqrtl	△ *1
math.h	acosl	△ *2			ceill	△
	asinl	△ *2			fabsl	△ *1
	atanl	△ *2			floorl	△
	atan2l	△ *2			fmodl	△
	cosl	△ *2			fpclassify	○
	sinl	△ *2			isfinite	○
	tanl	△ *2			isinf	○
	coshl	△			isnan	○
	sinhl	△			isnormal	○
	tanhl	△			signbit	○
	expl	△			isgreater	○
	frexpl	△			isgreaterequal	○
	ldexpl	△			isless	○
	logl	△			islessequal	○
	log10l	△			islessgreater	○
	modfl	△			isunordered	○
	powl	△			acosh	×

標準インクルードファイル	関数名	リエントラント	標準インクルードファイル	関数名	リエントラント
math.h	acoshf	×	math.h	cbrtf	○
	acoshl	×		cbrtl	○
	asinh	×		hypot	×
	asinhf	×		hypotf	×
	asinhl	×		hypotl	×
	atanh	×		erf	×
	atanhf	×		erff	×
	atanhl	×		erfl	×
	exp2	×		erfc	×
	exp2f	×		erfcf	×
	exp2l	×		erfcl	×
	expm1	△		lgamma	×
	expm1f	△		lgammaf	×
	expm1l	△		lgammal	×
	ilogb	○		tgamma	×
	ilogbf	○		tgammaf	×
	ilogbl	○		tgammal	×
	log1p	×		nearbyint	○
	log1pf	×		nearbyintf	○
	log1pl	×		nearbyintl	○
	log2	×		rint	×
	log2f	×		rintf	×
	log2l	×		rintl	×
	logb	×		lrint	×
	logbf	×		lrintf	×
	logbl	×		lrintl	×
	scalbn	×		llrint	×
	scalbnf	×		llrintf	×
	scalbnl	×		llrintl	×
	scalbln	×		round	○
	scalblnf	×		roundf	○
	scalblnl	×		roundl	○
	cbrt	○		lround	×

標準インクルードファイル	関数名	リエントラント	標準インクルードファイル	関数名	リエントラント
math.h	lroundf	×	math.h	fminf	○
	lroundl	×		fminl	○
	llround	×		fma	×
	llroundf	×		fmaf	×
	llroundl	×		fmal	×
	trunc	○	stdarg.h	va_copy	○
	truncf	○	stdio.h	snprintf	×
	truncl	○		vsnprintf	×
	remainder	×		vfscanf	×
	remainderf	×		vscanf	×
	remainderl	×		vsscanf	×
	remquo	×	complex.h	cacos	×
	remquof	×		cacosf	×
	remquol	×		cacosl	×
	copysign	○		casin	×
	copysignf	○		casinf	×
	copysignl	○		casinl	×
	nan	○		catan	×
	nanf	○		catanf	×
	nanl	○		catanl	×
	nextafter	×		ccos	×
	nextafterf	×		ccosf	×
	nextafterl	×		ccosl	×
	nexttoward	×		csin	×
	nexttowardf	×		csinf	×
	nexttowardl	×		csinl	×
	fdim	○		ctan	△
	fdimf	○		ctanf	△
	fdiml	○		ctanl	△
	fmax	○		cacosh	×
	fmaxf	○		cacoshf	×
	fmaxl	○		cacoshl	×
	fmin	○		casinh	×

標準インクルードファイル	関数名	リエントラント
complex.h	casinhf	×
	casinhl	×
	catanh	×
	catanhf	×
	catanhl	×
	ccosh	×
	ccoshf	×
	ccoshl	×
	csinh	×
	csinhf	×
	csinhl	×
	ctanh	△
	ctanhf	△
	ctanhl	△
	cexp	×
	cexpf	×
	cexpl	×
	clog	×
	clogf	×
	clogl	×
	cabs	×
	cabsf	×
	cabsl	×
	cpow	×
	cpowf	△
	cpowl	△
	csqrt	△
	csqrtf	△
	csqrtl	△
	carg	△
	cargf	△
	cargl	△
	cimag	○
complex.h	cimagf	○
	cimagl	○
	conj	○
	conjf	○
	conjl	○
	cproj	○
	cprojf	○
	cprojl	○
	creal	○
	crealf	○
	creall	○
fenv.h	feclearexcept	×
	fegetexceptflag	○
	feraiseexcept	×
	fesetexceptflag	×
	fetestexcept	○
	fegetround	○
	fesetround	×
	fegetenv	○
	feholdexcept	×
	fesetenv	×
	feupdateenv	×
inttypes.h	imaxabs	○
	imaxdiv	○
	strtoimax	△
	strtoumax	△
	wcstoimax	△
	wcstoumax	△
wchar.h	fwprintf	×
	vfwprintf	×
	swprintf	×
	vswprintf	×
	wprintf	×

標準インクルードファイル	関数名	リエントラント		標準インクルードファイル	関数名	リエントラント
wchar.h	vwprintf	×		wchar.h	wcstoull	△
	fwscanf	×			wcscpy	○
	vfwscanf	×			wcsncpy	○
	swscanf	×			wmemcpy	○
	vswscanf	×			wmemmove	○
	wscanf	×			wcscat	○
	vwscanf	×			wcsncat	○
	fgetwc	×			wcscmp	○
	fgetws	×			wcsncmp	○
	fputwc	×			wmemcmp	○
	fputws	×			wcschr	○
	fwide	×			wcscspn	○
	getwc	×			wcspbrk	○
	getwchar	×			wcsrchr	○
	putwc	×			wcsspn	○
	putwchar	×			wcsstr	○
	ungetwc	×			wcstok	○
	wcstod	×			wmemchr	○
	wcstof	×			wcslen	○
	wcstold	×			wmemset	○
	wcstol	△			mbsinit	○
	wcstoll	△			mbrlen	×
	wcstoul	○				

注 1. 関数呼び出しが命令に置き換わる場合は errno 変数を更新しないため ○ になります。
命令に置き換わる条件は、コンパイルオプションの -library の項目を参照ください。

注 2. 三角関数演算器を利用するコードに置き換わると ×(ノンリエントラント) になる場合があります。詳細は、コンパイルオプションの -tfu の項目を参照ください。

7.4 ライブラリ関数

この節では、ライブラリ関数について説明します。

CC-RX のライブラリ関数では、C99 言語で拡張されたキーワード (関数、マクロ、変数名) の一部に、C99 言語が選択された場合だけ使用できるものがあります。このような関数は、本項の各ヘッダファイルごとのキーワードを説明する表に、*<-lang=c99>* の表示を付記しております。これらの関数を使用する場合は、コンパイルおよびライブラリ生成時に、-lang=c99 オプションを指定してください。

7.4.1 <stddef.h>

標準インクルードファイルの中で共通に使用されるマクロ名を定義します。

以下は、すべて処理系定義です。

種別	定義名	説明
型 (typedef)	ptrdiff_t	2 つのポインタを減算した結果の型です。
	size_t	sizeof 演算子による演算結果の型です。
定数 (マクロ)	NULL	ポインタが何も指していない時の値です。 これは、0 と等値演算子 (==) による比較結果が真になるような値です。
変数 (マクロ)	errno	ライブラリ関数の処理中にエラーが発生した場合、そのライブラリごとに定義されたエラーコードがこの errno に設定されます。ライブラリ関数を呼び出す前に errno に 0 を設定しておき、ライブラリ関数の処理終了後に errno に設定されているコードを調べることによってライブラリ関数の処理中に発生したエラーをチェックすることができます。
関数 (マクロ)	offsetof	構造体メンバの構造体先頭からのオフセット値をバイト単位で求めます。
型 (typedef)	wchar_t	拡張文字を表す型です。

処理系定義仕様

	項目	
1	マクロ NULL の値	0 とします。
2	ptrdiff_t に適合する型	long 型
3	wchar_t に適合する型	unsigned short 型

7.4.2 <assert.h>

プログラム中に診断機能を付け加えます。

種別	定義名	説明
関数 (マクロ)	assert	プログラム中に診断機能を付け加えます。

<assert.h> で定義される診断機能を無効にするためには、<assert.h> を取り込む前に NDEBUG というマクロ名を #define 文で定義してください（#define NDEBUG）。

注 assert というマクロ名に対して #undef 文を使用すると、それ以降の assert の呼び出しの効果は保証しません。

assert

プログラム中に診断機能を付け加えます。

[指定形式]

```
#include <assert.h>
void assert(long expression);
```

[引数]

expression　評価する式

[備考]

assert マクロは、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。
処理系定義仕様 assert（expression）において、expression が偽の時メッセージを出力します。
なお、コンパイル時の lang オプションにより表示は変化します。

(1) -lang=c99 がないとき (C(C89)、C++、EC++ 言語の場合) :
ASSERTION FAILED:Δ 式 ΔFILEΔ< ファイル名 >,LINEΔ< 行番号 >

(2) -lang=c99 があるとき (C(C99) 言語の場合) :
ASSERTION FAILED:Δ 式 ΔFILEΔ< ファイル名 >,LINEΔ< 行番号 >ΔFUNCNAMEΔ< 関数名 >

7.4.3 <ctype.h>

文字に対して、その種類の判定や変換を行います。

種別	定義名	説明
関数	isalnum	英字または 10 進数字かどうかを判定します。
	isalpha	英字かどうかを判定します。
	iscntrl	制御文字かどうかを判定します。
	isdigit	10 進数字かどうかを判定します。
	isgraph	空白を除く印字文字かどうかを判定します。
	islower	英小文字かどうかを判定します。
	isprint	空白を含む印字文字かどうかを判定します。
	ispunct	特殊文字かどうかを判定します。
	isspace	空白類文字かどうかを判定します。
	isupper	英大文字かどうかを判定します。
	isxdigit	16 進数字かどうかを判定します。
	tolower	英大文字を英小文字に変換します。
	toupper	英小文字を英大文字に変換します。
	isblank <-lang=c99>	空白文字またはタブ文字かを判定します。

上記の関数において、入力パラメータの値が unsigned char 型で表現できる範囲に含まれず、なおかつ EOF でない場合、その関数の動作は保証しません。
文字の種類の一覧を表 7.6 に示します。

表 7.6 文字の種類

	文字の種類	内容
1	英大文字	以下の 26 文字のいずれかの文字です。 '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'
2	英小文字	以下の 26 文字のいずれかの文字です。 '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'
3	英字	英大文字と英小文字のいずれかの文字です。
4	10 進数字	以下の 10 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'
5	印字文字	空白 (' ') を含む、ディスプレイ上に表示される文字のことです。 ASCII コードの 0x20 ～ 0x7E に対応します。
6	制御文字	印字文字以外の文字のことです。
7	空白類文字	以下の 6 文字のいずれかの文字です。 空白 (' ')、書式送り ('\f')、改行 ('\n')、復帰 ('\r')、水平タブ ('\t')、垂直タブ ('\v')
8	16 進数字	以下の 22 文字のいずれかの文字です。 '0'、'1'、'2'、'3'、'4'、'5'、'6'、'7'、'8'、'9'、 'A'、'B'、'C'、'D'、'E'、'F'、'a'、'b'、'c'、'd'、'e'、'f'
9	特殊文字	空白 (' ')、英字、および 10 進数字を除く任意の印字文字のことです。

	文字の種類	内容
10	ブランク文字	以下の 2 文字のいずれかの文字です。 空白 (' ')、水平タブ ('\t')

処理系定義仕様

	項目	コンパイラの仕様
1	isalnum 関数、isalpha 関数、iscntrl 関数、islower 関数、isprint 関数および isupper 関数によってテストされる文字集合	unsigned char 型 (0 ～ 255) および EOF(-1) です。判定の結果、真になる文字を表 7.7 に示します。

表 7.7 真となる文字の集合

	関数名	真となる文字
1	isalnum	'0' ～ '9', 'A' ～ 'Z', 'a' ～ 'z'
2	isalpha	'A' ～ 'Z', 'a' ～ 'z'
3	iscntrl	'\x00' ～ '\x1f', '\x7f'
4	islower	'a' ～ 'z'
5	isprint	'\x20' ～ '\x7E'
6	isupper	'A' ～ 'Z'

isalnum

文字が英字または 10 進数字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isalnum(long c);
```

[引数]

c 判定する文字

[戻り値]

文字 c が英字または 10 進数字の時：0 以外
文字 c が英字または 10 進数字以外の時：0

isalpha

文字が英字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isalpha(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が英字の時：0 以外
文字 c が英字以外の時：0

iscntrl

文字が制御文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long iscntrl(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が制御文字の時：0 以外
文字 c が制御文字以外の時：0

isdigit

文字が 10 進数字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isdigit(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が 10 進数字の時：0 以外
文字 c が 10 進数字以外の時：0

isgraph

文字が空白 (' ') を除く任意の印字文字かどうかを判定します。

[指定形式]

```
#include <ctype.h>
long isgraph(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が空白を除く印字文字の時：0 以外
文字 c が空白を除く印字文字以外の時：0

islower

文字が英小文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long islower(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が英小文字の時：0 以外
文字 c が英小文字以外の時：0

isprint

文字が空白文字 (' ') を含む印字文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isprint(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が空白文字を含む印字文字の時：0 以外
文字 c が空白文字を含む印字文字以外の時：0

ispunct

文字が特殊文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long ispunct(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が特殊文字の時：0 以外
文字 c が特殊文字以外の時：0

isspace

文字が空白類文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isspace(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が空白類文字の時：0 以外
文字 c が空白類文字以外の時：0

isupper

文字が英大文字であるかどうか判定します。

[指定形式]

```
#include <ctype.h>
long isupper(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が英大文字の時：0 以外
文字 c が英大文字以外の時：0

isxdigit

文字が 16 進数字かどうか判定します。

[指定形式]

```
#include <ctype.h>
long isxdigit(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が 16 進数字の時：0 以外
文字 c が 16 進数字以外の時：0

tolower

英大文字を対応する英小文字に変換します。

[指定形式]

```
#include <ctype.h>
long tolower(long c);
```

[引数]

c　変換する文字

[戻り値]

文字 c が英大文字の時：文字 c に対応する英小文字
文字 c が英大文字以外の時：文字 c

toupper

英小文字を対応する英大文字に変換します。

[指定形式]

```
#include <ctype.h>
long toupper(long c);
```

[引数]

c　変換する文字

[戻り値]

文字 c が英小文字の時：文字 c に対応する英大文字
文字 c が英小文字以外の時：文字 c

isblank

空白文字またはタブ文字か判定します。

[指定形式]

```
#include <ctype.h>
long isblank(long c);
```

[引数]

c　判定する文字

[戻り値]

文字 c が空白文字またはタブ文字の時：0 以外
文字 c が空白文字でもタブ文字でもない時：0

7.4.4 <float.h>

浮動小数点型の内部表現に関する各種制限値を定義します。
以下はすべて処理系定義です。

種別	定義名	定義値	説明
定数 (マクロ)	FLT_RADIX	2	指数部表現における基数です。
	FLT_ROUNDS	1	演算結果を丸めるかどうかを示します。 1 は演算結果を丸めることを意味します。
	FLT_MAX	3.4028235677973364e+38F	float 型が浮動小数点数として表現できる最大値です。
	DBL_MAX	1.7976931348623158e+308	double 型が浮動小数点数として表現できる最大値です。
	LDBL_MAX	1.7976931348623158e+308	long double 型が浮動小数点数として表現できる最大値です。
	FLT_MAX_EXP	128	float 型が浮動小数点数として表現できる最大値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	DBL_MAX_EXP	1024	double 型が浮動小数点数として表現できる最大値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	LDBL_MAX_EXP	1024	long double 型が浮動小数点数として表現できる最大値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	FLT_MAX_10_EXP	38	float 型が正の値として表現できる最大の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。
	DBL_MAX_10_EXP	308	double 型が正の値として表現できる最大の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。
	LDBL_MAX_10_EXP	308	long double 型が正の値として表現できる最大の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。

種別	定義名	定義値	説明
定数 (マクロ)	FLT_MIN	1.175494351e-38F	float 型が浮動小数点数として表現できる正の値での最小値です。
	DBL_MIN	2.2250738585072014e-308	double 型が浮動小数点数として表現できる正の値での最小値です。
	LDBL_MIN	2.2250738585072014e-308	long double 型が浮動小数点数として表現できる正の値での最小値です。
	FLT_MIN_EXP	-125	float 型が正の値として表現できる最小値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	DBL_MIN_EXP	-1021	double 型が正の値として表現できる最小値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	LDBL_MIN_EXP	-1021	long double 型が正の値として表現できる最小値を、基数のべき乗を用いて表現したとき、その指数に 1 を加えた値です。
	FLT_MIN_10_EXP	-37	float 型が正の値として表現できる最小の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。
	DBL_MIN_10_EXP	-307	double 型が正の値として表現できる最小の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。
	LDBL_MIN_10_EXP	-307	long double 型が正の値として表現できる最小の整数値を、10 のべき乗を用いて表現したとき、その指数の値です。
	FLT_DIG	6	float型の浮動小数点値の10進精度の最大桁数です。
	DBL_DIG	15	double型の浮動小数点値の10進精度の最大桁数です。
	LDBL_DIG	15	long double型の浮動小数点値の10進精度の最大桁数です。
	FLT_MANT_DIG	24	float 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	DBL_MANT_DIG	53	double 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	LDBL_MANT_DIG	53	long double 型の浮動小数点値を基数に合わせて表現した時の仮数部の最大桁数です。
	DECIMAL_DIG	17	浮動小数点型数値の 10 進精度の最大桁数です。
	FLT_EPSILON	1.1920928955078125e-07	float型で表現可能な1より大きい最小の値と１との差を示します。
	DBL_EPSILON	2.2204460492503131e-16	double 型で表現可能な 1 より大きい最小の値と 1 との差を示します。
	LDBL_EPSILON	2.2204460492503131e-16	long double 型で表現可能な 1 より大きい最小の値と 1 との差を示します。

7.4.5 <limits.h>

整数型データの内部表現に関する各種制限値を定義します。
以下はすべて処理系定義です。

種別	定義名	定義値	説明
定数 (マクロ)	CHAR_BIT	8	char 型が何ビットから構成されるかを示します。
	CHAR_MAX	127	char 型の変数が値として持つことができる最大値です。
		255 *1	
	CHAR_MIN	-128	char 型の変数が値として持つことができる最小値です。
		0 *1	
	SCHAR_MAX	127	signed char 型の変数が値として持つことができる最大値です。
	SCHAR_MIN	-128	signed char 型の変数が値として持つことができる最小値です。
	UCHAR_MAX	255U	unsigned char 型の変数が値として持つことができる最大値です。
	SHRT_MAX	32767	short 型の変数が値として持つことができる最大値です。
	SHRT_MIN	-32768	short 型の変数が値として持つことができる最小値です。
	USHRT_MAX	65535U	unsigned short 型の変数が値として持つことができる最大値です。
	INT_MAX	2147483647	int 型の変数が値として持つことができる最大値です。
	INT_MIN	-2147483647-1	int 型の変数が値として持つことができる最小値です。
	UINT_MAX	4294967295U	unsigned int 型の変数が値として持つことができる最大値です。
	LONG_MAX	2147483647L	long 型の変数が値として持つことができる最大値です。
	LONG_MIN	-2147483647L-1L	long 型の変数が値として持つことができる最小値です。
	ULONG_MAX	4294967295U	unsigned long 型の変数が値として持つことができる最大値です。
	LLONG_MAX	92233720368547758O7LL	long long 型の変数が値として持つことができる最大値です。
	LLONG_MIN	-92233720368547758O7LL-1LL	long long 型の変数が値として持つことができる最小値です。
	ULLONG_MAX	18446744073709551615ULL	unsigned long long 型の変数が値として持つことができる最大値です。

注 1. signed_char オプションを指定した場合の変数が値として持つことができる値になります。

7.4.6 <errno.h>

ライブラリ関数においてエラーが発生したときに errno に設定する値を定義します。
以下は、すべて処理系定義です。

種別	定義名	説明
変数 (マクロ)	errno	int 型変数です。ライブラリ関数においてエラーが発生したときにエラー番号が設定されます。
定数 (マクロ)	ERANGE	「10.5.6 C 標準ライブラリ関数のエラーメッセージ」を参照してください。
	EDOM	
	ESTRN	
	PTRERR	
	ECBASE	
	ETLN	
	EEXP	
	EEXPN	
	EFLOATO	
	EFLOATU	
	EDBLO	
	EDBLU	
	ELDBLO	
	ELDBLU	
	NOTOPN	
	EBADF	
	ECSPEC	
	EFIXEDO	
	EFIXEDU	
	EACCUMO	
	EACCUMU	
	ELFIXEDO	
	ELFIXEDU	
	ELACCUMO	
	ELACCUMU	
	EILSEQ	

7.4.7 <math.h>

各種の数値計算を行います。
以下の定数 (マクロ) はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	EDOM	関数に入力するパラメータの値が関数内で定義している値の範囲を超える時、errno に設定する値です。
	ERANGE	関数の計算結果が double 型の値として表せない時、あるいはオーバフロー / アンダフローとなった時、errno に設定する値です。
	HUGE_VAL HUGE_VALF *<-lang=c99>* HUGE_VALL *<-lang=c99>*	関数の計算結果がオーバフローした時に、関数のリターン値として返す値です。
	INFINITY *<-lang=c99>*	正または符号なしの無限大を表す float 型の定数式に展開します。
	NAN *<-lang=c99>*	float 型の qNaN をサポートしている場合に定義されます。
	FP_INFINITE *<-lang=c99>* FP_NAN *<-lang=c99>* FP_NORMAL *<-lang=c99>* FP_SUBNORMAL *<-lang=c99>* FP_ZERO *<-lang=c99>*	浮動小数点数の値の排他的な種類を表します。
	FP_ILOGB0 *<-lang=c99>* FP_ILOGBNAN *<-lang=c99>*	それぞれ 0 または非数の場合に ilogb で返される値の整数定数式に展開します。
	MATH_ERRNO *<-lang=c99>* MATH_ERREXCEPT *<-lang=c99>*	それぞれ整数定数 1 および 2 に展開します。
	math_errhandling *<-lang=c99>*	Int 型で値が、MATH_ERRNO,MATH_ERREXCEPT のビット単位の論理和の式に展開します。
型	float_t *<-lang=c99>* double_t *<-lang=c99>*	それぞれ float 型、double 型と同じ幅を持つ浮動小数点型です。
関数 (マクロ)	fpclassify *<-lang=c99>*	実引数の値を非数、無限大、正規化数、非正規化数、0 に分類します。
	isfinite *<-lang=c99>*	実引数が有限の値か判定します。
	isinf *<-lang=c99>*	実引数が無限大か判定します。
	isnan *<-lang=c99>*	実引数が非数か判定します。
	isnormal *<-lang=c99>*	実引数が正規化数か判定します。
	signbit *<-lang=c99>*	実引数の符号が負か判定します。
	isgreater *<-lang=c99>*	最初の引数が２番目の引数より大きいかどうかを判定します。
	isgreaterequal *<-lang=c99>*	最初の引数が２番目の引数以上かどうかを判定します。
	isless *<-lang=c99>*	最初の引数が２番目の引数より小さいかどうかを判定します。
	islessequal *<-lang=c99>*	最初の引数が２番目の引数以下かどうかを判定します。
	islessgreater *<-lang=c99>*	最初の引数が２番目の引数より小さいまたは大きいを判定します。
	isunordered *<-lang=c99>*	順序付けられていないかどうかを判定します。

種別	定義名	説明
関数	acos / acosf / acosl	浮動小数点値の逆余弦を計算します。
	asin / asinf / asinl	浮動小数点値の逆正弦を計算します。
	atan / atanf / atanl	浮動小数点値の逆正接を計算します。
	atan2 / atan2f / atan2l	浮動小数点値どうしを除算した結果の値の逆正接を計算します。
	cos / cosf / cosl	浮動小数点値のラジアン値の余弦を計算します。
	sin / sinf / sinl	浮動小数点値のラジアン値の正弦を計算します。
	tan / tanf / tanl	浮動小数点値のラジアン値の正接を計算します。
	cosh / coshf / coshl	動小数点値の双曲線余弦を計算します。
	sinh / sinhf / sinhl	浮動小数点値の双曲線正弦を計算します。
	tanh / tanhf / tanhl	浮動小数点値の双曲線正接を計算します。
	exp / expf / expl	浮動小数点値の指数関数を計算します。
	frexp / frexpf / frexpl	浮動小数点値を [0.5,1.0] の値と 2 のべき乗の積に分解します。
	ldexp / ldexpf / ldexpl	浮動小数点値と 2 のべき乗の乗算を計算します。
	log / logf / logl	浮動小数点値の自然対数を計算します。
	log10 / log10f / log10l	浮動小数点値の 10 を底とする対数を計算します。
	modf / modff / modfl	浮動小数点値を整数部分と小数部分に分解します。
	pow / powf / powl	浮動小数点値のべき乗を計算します。
	sqrt / sqrtf / sqrtl	浮動小数点値の正の平方根を計算します。
	ceil / ceilf / ceill	浮動小数点値の小数点以下を切り上げた整数値を求めます。
	fabs / fabsf / fabsl	浮動小数点値の絶対値を計算します。

種別	定義名	説明
関数	floor / floorf / floorl	浮動小数点値の小数点以下を切り捨てた整数値を求めます。
	fmod / fmodf / fmodl	浮動小数点値どうしを除算した結果の余りを計算します。
	acosh / acoshf / acoshl <*-lang=c99*>	浮動小数点値の双曲線逆余弦を計算します。
	asinh / asinhf / asinhl <*-lang=c99*>	浮動小数点値の双曲線逆正弦を計算します。
	atanh / atanhf / atanhl <*-lang=c99*>	浮動小数点値の双曲線逆正接を計算します。
	exp2 / exp2f / exp2l <*-lang=c99*>	浮動小数点値の 2 の x 乗を計算します。
	expm1 / expm1f / expm1l <*-lang=c99*>	自然対数の x 乗から 1 を引いた値を計算します。
	ilogb / ilogbf / ilogbl <*-lang=c99*>	符号あり int の値として x の指数を抽出します。
	log1p / log1pf / log1pl <*-lang=c99*>	実引数に 1 を加えた値の自然対数を計算します。
	log2 / log2f / log2l <*-lang=c99*>	2 を底とする対数を計算します。
	logb / logbf / logbl <*-lang=c99*>	符号あり整数の値として x の指数を抽出します。
	scalbn / scalbnf / scalbnl / scalbln / scalblnf / scalblnl <*-lang=c99*>	X×FLT_RADIXn を計算します。
	cbrt / cbrtf / cbrtl <*-lang=c99*>	浮動小数点値の立方根を計算します。
	hypot / hypotf / hypotl <*-lang=c99*>	引数の 2 乗の和の平方根（$\sqrt{x^2 + y^2}$）を計算します。
	erf / erff / erfl <*-lang=c99*>	誤差関数を計算します。
	erfc / erfcf / erfcl <*-lang=c99*>	余誤差関数を計算します。
	lgamma / lgammaf / lgammal <*-lang=c99*>	ガンマ関数の絶対値の自然対数を計算します。
	tgamma / tgammaf / tgammal <*-lang=c99*>	ガンマ関数を計算します。
	nearbyint / nearbyintf / nearbyintl <*-lang=c99*>	浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。
	rint / rintf / rintl <*-lang=c99*>	nearbyint に対して、浮動小数点例外を生成することがあります。
	lrint / lrintf / lrintl / llrint / llrintf / llrintl <*-lang=c99*>	丸め方向に従って、最も近い整数値に丸めます。
	round / roundf / roundl <*-lang=c99*>	浮動小数点形式の最も近い整数値に丸めます。

種別	定義名	説明
関数	lround / lroundf / lroundl / llround / llroundf / llroundl <-lang=c99>	最も近い整数値に丸めます。
	trunc / truncf / truncl <-lang=c99>	浮動小数点形式の最も近い整数値に丸めます。
	remainder / remainderf / remainderl <-lang=c99>	IEEE60559 の剰余 x REM y を計算します。
	remquo / remquof / remquol <-lang=c99>	x/y と同符号で、商の絶対値を 2^n を法として合同である絶対値を計算します。
	copysign / copysignf / copysignl <-lang=c99>	絶対値、および符号が同じ値を生成します。
	nan / nanf / nanl <-lang=c99>	nan(“n 文字列 ”) は、strtod(“NAN(n 文字列)”, (char**) NULL) と等価です。
	nextafter / nextafterf / nextafterl <-lang=c99>	関数の型に変換して、実軸上の次に表現可能な値を求めます。
	nexttoward / nexttowardf / nexttowardl <-lang=c99>	2 番目の引数型が long double, 引数同士が等しい場合に、2 番目の引数をその関数の型に変換して返す以外は、nextafter 関数群と同じです。
	fdim / fdimf / fdiml <-lang=c99>	正の差を計算します。
	fmax / fmaxf / fmaxl <-lang=c99>	大きい方の値を求めます。
	fmin / fminf / fminl <-lang=c99>	小さいほうの値を求めます。
	fma / fmaf / fmal <-lang=c99>	(x×y)+z をひとつの 3 項演算としてまとめて計算します。

エラーが発生した時の動作を以下に説明します。

(1) 定義域エラー
関数に入力するパラメータの値が関数内で定義している値の範囲を超えている時、定義域エラーが発生します。この時 errno には EDOM の値が設定されます。また、関数のリターン値は、処理系定義です。

(2) 範囲エラー
関数における計算結果が double 型の値として表せない時には範囲エラーが発生します。この時、errno には ERANGE の値が設定されます。また、計算結果がオーバフローの時は、正しく計算が行われた時と同様の符号の HUGE_VAL、HUGE_VALF あるいは HUGE_VALL の値をリターン値として返します。逆に計算結果がアンダフローの時は、0 をリターン値として返します。

注 1. <math.h> の関数の呼び出しによって定義域エラーが発生する可能性がある場合は、結果の値をそのまま用いるのは危険です。必ず errno をチェックしてから用いてください。

例

```
.
.
.
1  x=asin(a);
2  if (errno==EDOM)
3  printf("error\n");
4  else
5  printf("result is : %lf\n",x);
.
.
.
```

1 行目で、asin 関数を使って逆正弦値を求めます。このとき、実引数 a の値が、asin 関数の定義域 [-1.0, 1.0] の範囲を超えていると、errno に値 EDOM が設定されます。2 行目で定義域エラーが生じたかどうかの判定をします。定義域エラーが生じれば、3 行目で、error を出力します。定義域エラーが生じなければ 5 行目で、逆正弦値を出力します。

注 2. 範囲エラーが発生するかどうかは、コンパイラによって定まる、浮動小数点型の内部表現形式によって異なります。例えば無限大を値として表現できる内部表現形式を採用している場合、範囲エラーの生じないように <math.h> のライブラリ関数を実現することができます。

処理系定義仕様

	項目	コンパイラの仕様
1	数学関数の入力実引数が範囲を超えたときの数学関数が返す値	非数を返します。非数の形式は「4.1.5 データの内部表現と領域」の「(5) 浮動小数点型の仕様」を参照してください。
2	数学関数でアンダフローエラーが発生したときマクロ「ERANGE」の値が「errno」に設定されるかどうか	errno に ERANGE を設定する関数については、「10.5.6 C 標準ライブラリ関数のエラーメッセージ」を参照してください。上記以外は設定しません。
3	fmod 関数で第 2 実引数の値が 0 の場合、範囲エラーとなるかどうか	範囲エラーとなります。 fmod の戻り値の詳細は、「7.4.7 <math.h>」の「fmod / fmodf / fmodl」を参照してください。

acos / acosf / acosl

浮動小数点値の逆余弦を計算します。

[指定形式]

```
#include <math.h>
double acos(double d);
float acosf(float d);
long double acosl(long double d);
```

[引数]

d　逆余弦を求める浮動小数点値

[戻り値]

正常：d の逆余弦値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が [-1.0,1.0] の範囲を超えている時、定義域エラーになります。
acos 関数のリターン値の範囲は [0,π] です。
三角関数演算器を利用するコードに置き換わるとリエントラントではなくなる場合があります。詳細は、コンパイル・オプションの -tfu の項目を参照ください。

asin / asinf / asinl

浮動小数点値の逆正弦を計算します。

[指定形式]

```
#include <math.h>
double asin(double d);
float asinf(float d);
long double asinl(long double);
```

[引数]

d　逆正弦を求める浮動小数点値

[戻り値]

正常：d の逆正弦値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が [-1.0,1.0] の範囲を超えている時、定義域エラーになります。
asin 関数のリターン値の範囲は [-π/2,π/2] です。
三角関数演算器を利用するコードに置き換わるとリエントラントではなくなる場合があります。詳細は、コンパイル・オプションの -tfu の項目を参照ください。

atan / atanf / atanl

浮動小数点値の逆正接を計算します。

[指定形式]

```
#include <math.h>
double atan(double d);
float atanf(float d);
long double atanl(long double d);
```

[引数]

d　逆正接を求める浮動小数点値

[戻り値]

d の逆正接値

[備考]

atan 関数のリターン値の範囲は (-π/2,π/2) です。
三角関数演算器を利用するコードに置き換わるとリエントラントではなくなる場合があります。詳細は、コンパイル・オプションの -tfu の項目を参照ください。

atan2 / atan2f / atan2l

浮動小数点値どうしを除算した結果の値の逆正接を計算します。

[指定形式]

```
#include <math.h>
double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

[引数]

x　除数
y　被除数

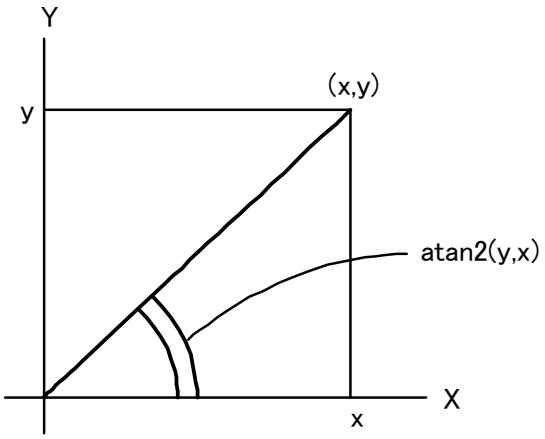
[戻り値]

正常：y を x で除算したときの逆正接値
異常：定義域エラーの時は、非数を返します

[備考]

x, y の値がともに 0.0 の時、定義域エラーになります。
atan2 関数のリターン値の範囲は (-π,π) です。atan2 関数の示す意味を図 7.1 に示します。図に示すように、atan2 関数の結果は、点 (x, y) と原点を通る直線と x 軸をなす角を求めます。y ＝ 0.0 で x が負の時、結果は π となります。
x ＝ 0.0 の時、y の値の正負に従って結果は ±π/2 となります。ただし、マイコンの設定によりゼロ除算例外が発生します。

図 7.1　atan2 関数の意味



cos / cosf / cosl

浮動小数点値のラジアン値の余弦を計算します。

[指定形式]

```
#include <math.h>
double cos(double d);
float cosf(float d);
long double cosl(long double d);
```

[引数]

d　余弦を求めるラジアン値

[戻り値]

d の余弦値

sin / sinf / sinl

浮動小数点値のラジアン値の正弦を計算します。

[指定形式]

```
#include <math.h>
double sin(double d);
float sinf(float d);
long double sinl(long double d);
```

[引数]

d　正弦を求めるラジアン値

[戻り値]

d の正弦値

tan / tanf / tanl

浮動小数点値のラジアン値の正接を計算します。

[指定形式]

```
#include <math.h>
double tan(double d);
float tanf(float d);
long double tanl(long double d);
```

[引数]

d　正接を求めるラジアン値

[戻り値]

d の正接値

[備考]

三角関数演算器を利用するコードに置き換わるとリエントラントではなくなる場合があります。詳細は、コンパイル・オプションの -tfu の項目を参照ください。

cosh / coshf / coshl

浮動小数点値の双曲線余弦を計算します。

[指定形式]

```
#include <math.h>
double cosh(double d);
float coshf(float d);
long double coshl(long double d);
```

[引数]

d　双曲線余弦を求める浮動小数点値

[戻り値]

d の双曲線正弦値

sinh / sinhf / sinhl

浮動小数点値の双曲線正弦を計算します。

[指定形式]

```
#include <math.h>
double sinh(double d);
float sinhf(float d);
long double sinhl(long double d);
```

[引数]

d　双曲線正弦を求める浮動小数点値

[戻り値]

d の双曲線正弦値

tanh / tanhf / tanhl

浮動小数点値の双曲線正接を計算します。

[指定形式]

```
#include <math.h>
double tanh(double d);
float tanhf(float d);
long double tanhl(long double d);
```

[引数]

d　双曲線正接を求める浮動小数点値

[戻り値]

d の双曲線正接値

exp / expf / expl

浮動小数点値の指数関数を計算します。

[指定形式]

```
#include <math.h>
double exp(double d);
float expf(float d);
long double expl(long double d);
```

[引数]

d　指数関数を求める浮動小数点値

[戻り値]

d の指数関数値

frexp / frexpf / frexpl

浮動小数点値を [0.5,1.0) の値と 2 のべき乗の積に分解します。

[指定形式]

```
#include <math.h>
double frexp(double value, long *exp);
float frexpf(float value, long *exp);
long double frexpl(long double value, long *exp);
```

[引数]

value　[0.5,1.0) の値と 2 のべき乗の積に分解する浮動小数点値
exp　2 のべき乗値を格納する記憶域へのポインタ

[戻り値]

value が 0.0 の時：0.0
value が 0.0 でない時：ret*2exp の指している領域の値 =value で定義される ret の値

[備考]

frexp 関数は、value を [0.5,1.0) の値と 2 のべき乗の積に分解します。exp の指す領域には、分解した結果の 2 のべき乗値を設定します。
リターン値 ret の値の範囲は [0.5,1.0) または 0.0 になります。
value が 0.0 ならば、exp の指す int 型の記憶域の内容と ret の値は 0.0 になります。

ldexp / ldexpf / ldexpl

浮動小数点値と 2 のべき乗の積を計算します。

[指定形式]

```
#include <math.h>
double ldexp(double e, long f);
float ldexpf(float e, long f);
long double ldexpl(long double e, long f);
```

[引数]

e　2 のべき乗値を求める浮動小数点値
f　2 のべき乗値

[戻り値]

$e*2^f$ の演算結果の値

log / logf / logl

浮動小数点値の自然対数を計算します。

[指定形式]

```
#include <math.h>
double log(double d);
float logf(float d);
long double logl(long double d);
```

[引数]

d　自然対数を求める浮動小数点値

[戻り値]

正常：d の自然対数の値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が負の値の時、定義域エラーになります。
d の値が 0.0 の時、範囲エラーになります。

log10 / log10f / log10l

浮動小数点値の 10 を底とする対数を計算します。

[指定形式]

```
#include <math.h>
double log10(double d);
float log10f(float d);
long double log10l(long double d);
```

[引数]

d　10 を底とする対数を求める浮動小数点値

[戻り値]

正常：d は 10 を底とする対数値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が負の値の時、定義域エラーになります。
d の値が 0.0 の時、範囲エラーになります。

modf / modff / modfl

浮動小数点値を整数部分と小数部分に分解します。

[指定形式]

```
#include <math.h>
double modf(double a, double *b);
float modff(float a, float *b);
long double modfl(long double a, long double *b);
```

[引数]

a 整数部分と小数部分に分解する浮動小数点値
b 整数部分を格納する記憶域を指すポインタ

[戻り値]

a の小数部分

pow / powf / powl

浮動小数点値のべき乗を計算します。

[指定形式]

```
#include <math.h>
double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

[引数]

x　べき乗される値
y　べき乗する値

[戻り値]

正常：x の y 乗の値
異常：定義域エラーの時は、非数を返します

[備考]

エラー条件 x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。

sqrt / sqrtf / sqrtl

浮動小数点値の正の平方根を計算します。

[指定形式]

```
#include <math.h>
double sqrt(double d);
float sqrtf(float d);
long double sqrtl(long double d);
```

[引数]

d　正の平方根を求める浮動小数点値

[戻り値]

正常：d の正の平方根の値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が負の値の時、定義域エラーになります。

ceil / ceilf / ceill

浮動小数点値の小数点以下を切り上げた整数値を求めます。

[指定形式]

```
#include <math.h>
double ceil(double d);
float ceilf(float d);
long double ceill(long double d);
```

[引数]

d　小数点以下を切り上げる浮動小数点値

[戻り値]

d の小数点以下を切り上げた整数値

[備考]

ceil 関数は、d の値より大きいかまたは等しい最小の整数値を double 型の値として返す関数です。したがって d の値が負の値の時は小数点以下を切り捨てた時の値を返します。

fabs / fabsf / fabsl

浮動小数点値の絶対値を計算します。

[指定形式]

```
#include <math.h>
double fabs(double d);
float fabsf(float d);
long double fabsl(long double d);
```

[引数]

d　絶対値を求める浮動小数点値

[戻り値]

d の絶対値

floor / floorf / floorl

浮動小数点値の小数点以下を切り捨てた整数値を求めます。

[指定形式]

```
#include <math.h>
double floor(double d);
float floorf(float d);
long double floorl(long double d);
```

[引数]

d　小数点以下を切り捨てる浮動小数点値

[戻り値]

d の小数点以下を切り捨てた整数値

[備考]

floor 関数は、d の値を超えない範囲の整数の最大値を、double 型の値として返す関数です。したがって d の値が負の値の時は小数点以下を切り上げた時の値を返します。

fmod / fmodf / fmodl

浮動小数点値どうしを除算した結果の余りを計算します。

[指定形式]

```
#include <math.h>
double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

[引数]

x 被除数
y 除数

[戻り値]

y の値が 0.0 の時：x
y の値が 0.0 でない時：x を y で除算した結果の余り
y が ± ∞の場合、x を返します。
x が ± ∞の場合、または y が 0 の場合、非数を返し、グローバル変数 errno にマクロ EDOM を設定します。

[備考]

fmod 関数では、引数 x、y、リターン値 ret の間には、次に示す関係が成立します。
x=y*i+ret （ただし i は整数値）
また、リターン値 ret の符号は x の符号と同じ符号になります。x/y の商を表現できない場合、結果の値は保証しません。

acosh / acoshf / acoshl

双曲線逆余弦を計算します。

[指定形式]

```
#include <math.h>
double acosh(double d);
float acoshf(float d);
long double acoshl(long double d);
```

[引数]

d　双曲線逆余弦を求める浮動小数点値

[戻り値]

正常：d の双曲線逆余弦値
異常：定義域エラーの時は、非数を返します

[備考]

d の値が 1.0 未満の時、定義域エラーになります。
acosh 関数のリターン値の範囲は [0,+ ∞] です。

asinh / asinhf / asinhl

双曲線逆正弦を計算します。

[指定形式]

```
#include <math.h>
double asinh(double d);
float asinhf(float d);
long double asinhl(long double d);
```

[引数]

d　双曲線逆正弦を求める浮動小数点値

[戻り値]

d の双曲線逆正弦値

atanh / atanhf / atanhl

双曲線逆正接を計算します。

[指定形式]

```
#include <math.h>
double atanh(double d);
float atanhf(float d);
long double atanhl(long double d);
```

[引数]

d　双曲線逆正接を求める浮動小数点値

[戻り値]

正常：d の双曲線逆正接値
異常：定義域エラーの場合は関数に応じて HUGE_VAL, HUGE_VALF, HUGE_VALL
　　範囲エラーの場合は非数

[備考]

d の値が [-1,+1] の範囲にない場合は定義域エラーになります。
d の値が -1 または 1 に等しい場合、範囲エラーになる可能性があります。

exp2 / exp2f / exp2l

2 の d 乗を計算します。

[指定形式]

```
#include <math.h>
double exp2(double d);
float exp2f(float d);
long double exp2l(long double d);
```

[引数]

d　指数関数を求める浮動小数点数

[戻り値]

正常：2 の指数関数値
異常：範囲エラーの場合は 0 又は関数に応じて +HUGE_VAL, +HUGE_VALF, +HUGE_VALL

[備考]

d の絶対値が大きすぎる場合、範囲エラーになります。

expm1 / expm1f / expm1l

自然対数の底 e の d 乗から 1 を引いた値を計算します。

[指定形式]

```
#include <math.h>
double expm1(double d);
float expm1f(float d);
long double expm1l(long double d);
```

[引数]

d　自然対数の底 e の指数となる値

[戻り値]

正常：自然対数の底 e の d 乗から 1 を引いた値
異常：範囲エラーの場合は関数に応じて -HUGE_VAL, -HUGE_VALF, -HUGE_VALL

[備考]

d の値が 0 に近い場合でも expm1(d) は exp(x)-1 よりも正確に計算できます。

ilogb / ilogbf / ilogbl

d の指数を抽出します。

[指定形式]

```
#include <math.h>
long ilogb(double d);
long ilogbf(float d);
long ilogbl(long double d);
```

[引数]

d　指数を抽出する値

[戻り値]

正常：d の指数関数値
　d が∞の場合は INT_MAX
　d が非数の場合は FP_ILOGBNAN
　d が 0 の場合は FP_ILOGBNAN
異常：d が 0 で範囲エラーの場合は FP_ILOGB0

[備考]

d の値が 0 の場合、範囲エラーになることがあります。

log1p / log1pf / log1pl

d に 1 を加えた値の e を底とする自然対数を計算します。

[指定形式]

```
#include <math.h>
double log1p(double d);
float log1pf(float d);
long double log1pl(long double d);
```

[引数]

d　引数に 1 を加えた値の自然対数を計算する値

[戻り値]

正常：d に 1 を加えた値の自然対数
異常：定義域エラーの場合は非数
　　範囲エラーの場合は関数に応じて -HUGE_VAL, -HUGE_VALF, -HUGE_VALL

[備考]

d の値が -1 より小さい場合、定義域エラーになります。
d の値が -1 の場合、範囲エラーになります。
d の値が 0 に近い場合でも log1p(d) は log(1+d) より正確な計算ができます。

log2 / log2f / log2l

d の 2 を底とする対数を計算します。

[指定形式]

```
#include <math.h>
double log2(double d);
float log2f(float d);
long double log2l(long double d);
```

[引数]

d　対数を計算する値

[戻り値]

正常：d の 2 を底とする対数
異常：定義域エラーの場合は非数

[備考]

d の値が負の値の場合、定義域エラーになります。

logb / logbf / logbl

d の浮動小数点数の内部表現における指数部を浮動小数点値として抽出します。

[指定形式]

```
#include <math.h>
double logb(double d);
float logbf(float d);
long double logbl(long double d);
```

[引数]

d　指数を抽出する値

[戻り値]

正常：d の符号付き指数
異常：範囲エラーの場合は関数に応じて -HUGE_VAL, -HUGE_VALF, -HUGE_VALL

[備考]

d の値が 0 の場合、範囲エラーになることがあります。
d は常に正規化されているものとして処理します。

scalbn / scalbnf / scalbnl / scalbln / scalblnf / scalblnl

浮動小数点数に整数である基数の累乗を計算します。

[指定形式]

```
#include <math.h>
double scalbn(double d, long e);
float scalbnf(float d, long e);
long double scalbnl(long double d, long e);
double scalbln(double d, long e);
float scalblnf(float d, long int e);
long double scalblnl(long double d, long int e);
```

[引数]

d　FLT_RADIX を e 乗した値と乗算する値
e　FLT_RADIX を累乗する際に指数となる値

[戻り値]

正常：d と FLT_RADIX を乗算した値と等価の値
異常：範囲エラーの場合は関数に応じて -HUGE_VAL, -HUGE_VALF, -HUGE_VALL

[備考]

d の値が 0 の場合、範囲エラーになることがあります。
実際に e を指数とした FLT_RADIX の累乗は計算しません。

cbrt / cbrtf / cbrtl

浮動小数点値の立方根を計算します。

[指定形式]

```
#include <math.h>
double cbrt(double d);
float cbrtf(float d);
long double cbrtl(long double d);
```

[引数]

d　立方根を求める値

[戻り値]

d の立方根値

hypot / hypotf / hypotl

浮動小数点値の 2 乗の和の平方根（$\sqrt{x^2 + y^2}$）を計算します。

[指定形式]

```
#include <math.h>
double hypot(double x, double y);
float hypotf(float x, float y);
long double hypotl(long double x, long double y);
```

[引数]

x, y　2 乗の和の平方根（$\sqrt{x^2 + y^2}$）を求める値

[戻り値]

正常：2 乗の和の平方根（$\sqrt{x^2 + y^2}$）の値
異常：範囲エラーの場合は関数に応じて HUGE_VAL, HUGE_VALF, HUGE_VALL

[備考]

結果がオーバーフローする場合に範囲エラーになることがあります。

erf / erff / erfl

浮動小数点値の誤差関数を計算します。

[指定形式]

```
#include <math.h>
double erf(double d);
float erff(float d);
long double erfl(long double d);
```

[引数]

d　誤差関数値を求める値

[戻り値]

d の誤差関数値

erfc / erfcf / erfcl

浮動小数点値の余誤関数を計算します。

[指定形式]

```
#include <math.h>
double erfc(double d);
float erfcf(float d);
long double erfcl(long double d);
```

[引数]

d　余誤関数値を求める値

[戻り値]

d の余誤関数値

[備考]

d の絶対値が大きすぎる場合、範囲エラーが発生します。

lgamma / lgammaf / lgammal

浮動小数点値のガンマ関数の対数を計算します。

[指定形式]

```
#include <math.h>
double lgamma(double d);
float lgammaf(float d);
long double lgammal(long double d);
```

[引数]

d　ガンマ関数の対数値を求める値

[戻り値]

正常な場合：d のガンマ関数の対数値
定義域エラーの場合：数学的に正しい符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL
範囲エラーの場合：+HUGE_VAL, +HUGE_VALF, +HUGE_VALL

[備考]

d の絶対値が大きすぎる又は小さすぎる場合、範囲エラーを設定します。
d の値が負の整数又は 0 で、計算結果が表現できない場合、定義域エラーが発生します。

tgamma / tgammaf / tgammal

浮動小数点値のガンマ関数を計算します。

[指定形式]

```
#include <math.h>
double tgamma(double d);
float tgammaf(float d);
long double tgammal(long double d);
```

[引数]

d　ガンマ関数値を求める値

[戻り値]

正常な場合：d のガンマ関数値
定義域エラーの場合：d と同じ符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL
範囲エラーの場合：0 又は関数に応じて数学的に正しい符号が付与された +HUGE_VAL, +HUGE_VALF, +HUGE_VALL

[備考]

d の絶対値が大きすぎる又は小さすぎる場合、範囲エラーを設定します。
d の値が負の整数又は 0 で、計算結果が表現できない場合、定義域エラーが発生します。

nearbyint / nearbyintf / nearbyintl

浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。

[指定形式]

```
#include <math.h>
double nearbyint(double d);
float nearbyintf(float d);
long double nearbyintl(long double d);
```

[引数]

d　浮動小数点形式の整数値に丸める値

[戻り値]

d の浮動小数点形式の整数値に丸めた値

[備考]

nearbyint 関数群は ” 不正確結果 ” 浮動小数点例外を生成しません。

rint / rintf / rintl

浮動小数点値を丸め方向にしたがって、浮動小数点形式の整数値に丸めます。

[指定形式]

```
#include <math.h>
double rint(double d);
float rintf(float d);
long double rintl(long double d);
```

[引数]

d　浮動小数点形式の整数値に丸める値

[戻り値]

d の浮動小数点形式の整数値に丸めた値

[備考]

rint 関数群は " 不正確結果 " 浮動小数点例外を生成する場合があるという点のみで nearbyint 関数群と違います。

lrint / lrintf / lrintl / llrint / llrintf / llrintl

浮動小数点値を丸め方向にしたがって、最も近い整数値に丸めます。

[指定形式]

```
#include <math.h>
long int lrint (double d);
long int lrintf(float d);
long int lrintl(long double d);
long long int llrint (double d);
long long int llrintf(float d);
long long int llrintl(long double d);
```

[引数]

d　整数に丸める値

[戻り値]

正常：d を整数値に丸めた値
異常：範囲エラーの場合は不定

[備考]

d の絶対値が大きすぎる場合、範囲エラーが発生する場合があります。
丸めた値がリターン値型の範囲外である場合のリターン値は未規定とします。

round / roundf / roundl

浮動小数点値を最も近い整数値に丸めます。

[指定形式]

```
#include <math.h>
double round(double d);
float roundf(float d);
long double roundl(long double d);
```

[引数]

d　整数に丸める値

[戻り値]

正常：d を整数値に丸めた値
異常：範囲エラーの場合は不定

[備考]

d の絶対値が大きすぎる場合、範囲エラーが発生する場合があります。

lround / lroundf / lroundl / llround / llroundf / llroundl

最も近い整数値に丸めます。

[指定形式]

```
#include <math.h>
long int lround(double d);
long int lroundf(float d);
long int lroundl(long double d);
long long int llround (double d);
long long int llroundf(float d);
long long int llroundl(long double d);
```

[引数]

d　整数に丸める値

[戻り値]

正常：d を整数値に丸めた値
異常：範囲エラーの場合は不定

[備考]

d の絶対値が大きすぎる場合、範囲エラーが発生する場合があります。
lround 関数群は d の値が中間にある場合、その時点の丸め方向とは関係なく 0 から遠い方向を選んで丸めます。丸めた値がリターン値型の範囲外である場合のリターン値は未規定とします。

trunc / truncf / truncl

浮動小数点値を最も近い浮動小数点形式の整数値に丸めます。

[指定形式]

```
#include <math.h>
double trunc(double d);
float truncf(float d);
long double truncl(long double d);
```

[引数]

d　浮動小数点形式の整数に丸める値

[戻り値]

d を切り捨てた浮動小数点形式の整数値

[備考]

trunc 関数群は丸めた値の絶対値が d の絶対値より大きくならないようにします。

remainder / remainderf / remainderl

浮動小数点数同士の剰余を計算します。

[指定形式]

```
#include <math.h>
double remainder(double d1, double d2);
float remainderf(float d1, float d2);
long double remainderl(long double d1, long double d2);
```

[引数]

d1, d2　剰余を求める値 (d1 / d2)

[戻り値]

d1 と d2 の剰余値

[備考]

remainder 関数群の剰余計算は IEEE 60559 の規定に沿っています。

remquo / remquof / remquol

浮動小数点値を最も近い整数値に丸めます。

[指定形式]

```
#include <math.h>
double remquo(double d1, double d2, long *q);
float remquof(float d1, float d2, long *q);
long double remquol(long double d1, long double d2, long *q);
```

[引数]

d1, d2　整数に丸める値 (d1 / d2)
q　剰余計算結果の商を格納する場所を指す値

[戻り値]

d1 と d2 の剰余値

[備考]

q に格納される値は x/y と同じ符号と、2n(n は 3 以上の処理系定義整数値) を法とする x/y の整数の商を持ちます。

copysign / copysignf / copysignl

絶対値が d1 に等しく、符号ビットが d2 に等しい値を生成します。

[指定形式]

```
#include <math.h>
double copysign(double d1, double d2);
float copysignf(float d1, float d2);
long double copysignl(long double d1, long double d2);
```

[引数]

d1　生成する絶対値の値
d2　生成する符号

[戻り値]

正常：d1 の絶対値、d2 の符号の値
異常：範囲エラーの場合は不定

[備考]

copysign 関数群は d1 が非数の場合 d2 の符号ビットを持った非数を生成します。

nan / nanf / nanl

非数を返します。

[指定形式]

```
#include <math.h>
double nan(const char *c);
float nanf(const char *c);
long double nanl(const char *c);
```

[引数]

c　文字列ポインタ

[戻り値]

c が示す内容をもつ qNaN または 0(qNaN 未サポート時)

[備考]

nan("c 文字列 ") の呼出しは、strtod("NAN(c 文字列)", (char**) NULL) と等価です。nanf 及び nanl の呼出しは、strtof 及び strtold のそれぞれに対応する呼出しと等価です。

nextafter / nextafterf / nextafterl

実軸上で d1 から見て d2 に向かう方向で d1 のすぐ次の浮動小数点数表現を計算します。

[指定形式]

```
#include <math.h>
double nextafter(double d1, double d2);
float nextafterf(float d1, float d2);
long double nextafterl(long double d1, long double d2);
```

[引数]

d1　実軸上の浮動小数点値
d2　d1 から見て表現可能な浮動小数点値の存在する方向を示す値

[戻り値]

正常：表現可能な浮動小数点値
異常：範囲エラーの場合、関数に応じて数学的に正しい符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL

[備考]

d1 がその型で表現できる最大の有限な値であり、かつリターン値が無限大又はその型で表現できない場合、範囲エラーが発生することがあります。
nextafter 関数群は d1 と d2 が等しい場合、d2 を返します。

nexttoward / nexttowardf / nexttowardl

実軸上で d1 から見て d2 に向かう方向で d1 のすぐ次の浮動小数点数表現を計算します。

[指定形式]

```
#include <math.h>
double nexttoward(double d1, long double d2);
float nexttowardf(float d1, long double d2);
long double nexttowardl(long double d1, long double d2);
```

[引数]

d1　実軸上の浮動小数点値
d2　d1 から見て表現可能な浮動小数点値の存在する方向を示す値

[戻り値]

正常：表現可能な浮動小数点値
異常：範囲エラーの場合、関数に応じて数学的に正しい符号が付与された HUGE_VAL, HUGE_VALF, HUGE_VALL

[備考]

d1 がその型で表現できる最大の有限な値であり、かつリターン値が無限大又はその型で表現できない場合、範囲エラーが発生することがあります。
nexttoward 関数群は d2 の値が long double であり、d1 と d2 が等しい場合は d2 を関数に応じて変換して返すという点以外は nextafter 関数群と等価です。

fdim / fdimf / fdiml

２引数間の正の差分を求めます。

[指定形式]

```
#include <math.h>
double fdim(double d1, double d2);
float fdimf(float d1, float d2);
long double fdiml(long double d1, long double d2);
```

[引数]

d1, d2　正の差を求める値 (|d1 - d2|)

[戻り値]

正常：２引数間の正の差分
異常：範囲エラーの場合は HUGE_VAL, HUGE_VALF, HUGE_VALL

[備考]

リターン値がオーバーフローした場合に範囲エラーが発生することがあります。

fmax / fmaxf / fmaxl

２引数の大きい方を求めます。

[指定形式]

```
#include <math.h>
double fmax(double d1, double d2);
float fmaxf(float d1, float d2);
long double fmaxl(long double d1, long double d2);
```

[引数]

d1, d2　大きさを比較する値

[戻り値]

２引数の大きい方

[備考]

fmax 関数群は非数を、データが欠けているものとして認識します。一方の引数が非数で、もう一方が数値の場合、数値の値を返します。

fmin / fminf / fminl

２引数の小さい方を求めます。

[指定形式]

```
#include <math.h>
double fmin(double d1, double d2);
float fminf(float d1, float d2);
long double fminl(long double d1, long double d2);
```

[引数]

d1, d2　大きさを比較する値

[戻り値]

２引数の小さい方

[備考]

fmin 関数群は非数を、データが欠けているものとして認識します。一方の引数が非数で、もう一方が数値の場合、数値の値を返します。

fma / fmaf / fmal

(d1*d2)+d3 を一つの 3 項演算としてまとめて計算します。

[指定形式]

```
#include <math.h>
double fma(double d1, double d2, double d3);
float fmaf(float d1, float d2, float d3);
long double fmal(long double d1, long double d2, long double d3);
```

[引数]

d1, d2, d3　浮動小数点値

[戻り値]

(d1*d2)+d3 を 3 項演算としてまとめて計算した結果

[備考]

fma 関数群は計算結果を無限の精度であるものとして計算し、FLT_ROUNDS の値が示す丸めモードに従って、1 回だけ丸めます。

7.4.8 <mathf.h>

各種の数値計算を行います。
<mathf.h> では ANSI 規格規定外の単精度形式の数学関数の宣言とマクロの定義をしています。
各関数は float 型の実引数を受け取り、float 型の値を返します。

以下の定数 (マクロ) はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	EDOM	関数に入力するパラメータの値が関数内で定義している値の範囲を超える時、errno に設定する値です。
	ERANGE	関数の計算結果が float 型の値として表せない時、あるいはオーバフロー / アンダフローとなった時、errno に設定する値です。
	HUGE_VALF HUGE_VAL HUGE_VALL	関数の計算結果がオーバフローした時に、関数のリターン値として返す値です。
関数	acosf	浮動小数点値の逆余弦を計算します。
	asinf	浮動小数点値の逆正弦を計算します。
	atanf	浮動小数点値の逆正接を計算します。
	atan2f	浮動小数点値どうしを除算した結果の値の逆正接を計算します。
	cosf	浮動小数点値のラジアン値の余弦を計算します。
	sinf	浮動小数点値のラジアン値の正弦を計算します。
	tanf	浮動小数点値のラジアン値の正接を計算します。
	coshf	浮動小数点値の双曲線余弦を計算します。
	sinhf	浮動小数点値の双曲線正弦を計算します。
	tanhf	浮動小数点値の双曲線正接を計算します。
	expf	浮動小数点値の指数関数を計算します。
	frexpf	浮動小数点値を [0.5, 1.0) の値と 2 のべき乗の積に分解します。
	ldexpf	浮動小数点値と 2 のべき乗の乗算を計算します。
	logf	浮動小数点値の自然対数を計算します。
	log10f	浮動小数点値の 10 を底とする対数を計算します。
	modff	浮動小数点値を整数部分と小数部分に分解します。
	powf	浮動小数点値のべき乗を計算します。
	sqrtf	浮動小数点値の正の平方根を計算します。
	ceilf	浮動小数点値の小数点以下を切り上げた整数値を求めます。
	fabsf	浮動小数点値の絶対値を計算します。
	floorf	浮動小数点値の小数点以下を切り捨てた整数値を求めます。
	fmodf	浮動小数点値どうしを除算した結果の余りを計算します。

エラーが発生した時の動作を以下に説明します。

(1) 定義域エラー
関数に入力するパラメータの値が関数内で定義している値の範囲を超えている時、定義域エラーが発生します。この時 errno には EDOM の値が設定されます。また、関数のリターン値は、処理系定義です。

(2) 範囲エラー
関数における計算結果が float 型の値として表せない時には範囲エラーが発生します。この時、errno には ERANGE の値が設定されます。また、計算結果がオーバフローの時は、正しく計算が行われた時と同様の符号の値をリターン値として返します。逆に計算結果がアンダフローの時は、0 をリターン値として返します。

注 1. <mathf.h> の関数の呼び出しによって定義域エラーが発生する可能性がある場合は、結果の値をそのまま用いるのは危険です。必ず errno をチェックしてから用いてください。

例

```
.
.
.
1  x=asinf(a);
2  if (errno==EDOM)
3  printf("error\n");
4  else
5  printf("result is : %f\n",x);
.
.
.
```

1 行目で、asinf 関数を使って逆正弦値を求めます。このとき、実引数 a の値が、asinf 関数の定義域 [-1.0,1.0] の範囲を超えていると、errno に値 EDOM が設定されます。2 行目で定義域エラーが生じたかどうかの判定をします。定義域エラーが生じれば、3 行目で、error を出力します。定義域エラーが生じなければ 5 行目で、逆正弦値を出力します。

注 2. 範囲エラーが発生するかどうかは、コンパイラによって定まる、浮動小数点型の内部表現形式によって異なります。例えば無限大を値として表現できる内部表現形式を採用している場合、範囲エラーの生じないように <mathf.h> のライブラリ関数を実現することができます。

処理系定義仕様

	項目	コンパイラの仕様
1	数学関数の入力実引数が範囲を超えたときの数学関数が返す値	非数を返します。非数の形式は「4.1.5 (5) 浮動小数点型の仕様」を参照してください
2	数学関数でアンダフローエラーが発生したときマクロ「ERANGE」の値が「errno」に設定されるかどうか	errno に ERANGE を設定する関数については、「10.5.6 C 標準ライブラリ関数のエラーメッセージ」を参照してください。 上記以外は設定しません。
3	fmodf 関数で第 2 実引数の値が 0 の場合、範囲エラーとなるかどうか	範囲エラーとなります。 fmod の戻り値の詳細は、7.4.7 <math.h> fmod / fmodf / fmodl を参照してください。

acosf

浮動小数点値の逆余弦を計算します。

[指定形式]

```
#include <mathf.h>
float acosf(float f);
```

[引数]

f　逆余弦を求める浮動小数点値

[戻り値]

正常：f の逆余弦値
異常：定義域エラーの時は、非数を返します

[備考]

f の値が [-1.0,1.0] の範囲を超えている時、定義域エラーになります。
acosf 関数のリターン値の範囲は [0,π] です。

asinf

浮動小数点値の逆正弦を計算します。

[指定形式]

```
#include <mathf.h>
float asinf(float f);
```

[引数]

f　逆正弦を求める浮動小数点値

[戻り値]

正常：f の逆正弦値
異常：定義域エラーの時は、非数を返します

[備考]

f の値が [-1.0,1.0] の範囲を超えている時、定義域エラーになります。
asinf 関数のリターン値の範囲は [-π/2,π/2] です。

atanf

浮動小数点値の逆正接を計算します。

[指定形式]

```
#include <mathf.h>
float atanf(float f);
```

[引数]

f　逆正接を求める浮動小数点値

[戻り値]

f の逆正接値

[備考]

atanf 関数のリターン値の範囲は (-π/2,π/2) です。

atan2f

浮動小数点値どうしを除算した結果の値の逆正接を計算します。

[指定形式]

```
#include <mathf.h>
float atan2f(float y, float x);
```

[引数]

x 除数
y 被除数

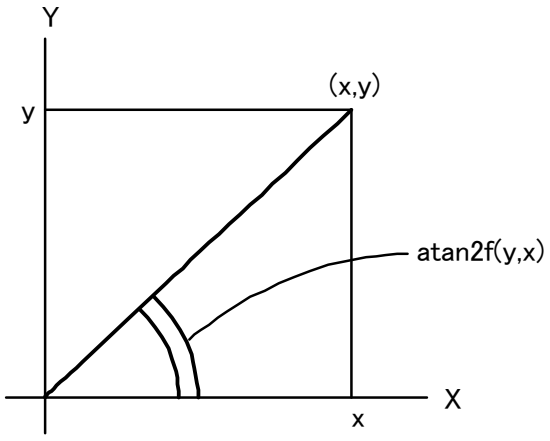
[戻り値]

正常：y を x で除算したときの逆正接値
異常：定義域エラーの時は、非数を返します

[備考]

x, y の値がともに 0.0 の時、定義域エラーになります。
atan2f 関数のリターン値の範囲は (-π,π) です。atan2f 関数の示す意味を図 7.2 に示します。図に示すように、atan2f 関数の結果は、点 (x,y) と原点を通る直線と x 軸をなす角を求めます。y ＝ 0.0 で x が負の時、結果は π、x ＝ 0.0 の時、y の値の正負に従って結果は ±π/2 となります。

図 7.2 atan2f 関数の意味



cosf

浮動小数点値のラジアン値の余弦を計算します。

[指定形式]

```
#include <mathf.h>
float cosf(float f);
```

[引数]

f　余弦を求めるラジアン値

[戻り値]

f の余弦値

sinf

浮動小数点値のラジアン値の正弦を計算します。

[指定形式]

```
#include <mathf.h>
float sinf(float f);
```

[引数]

f　正弦を求めるラジアン値

[戻り値]

f の正弦値

tanf

浮動小数点値のラジアン値の正接を計算します。

[指定形式]

```
#include <mathf.h>
float tanf(float f);
```

[引数]

f　正接を求めるラジアン値

[戻り値]

f の正接値

coshf

浮動小数点値の双曲線余弦を計算します。

[指定形式]

```
#include <mathf.h>
float coshf(float f);
```

[引数]

f　双曲線余弦を求める浮動小数点値

[戻り値]

f の双曲線余弦値

sinhf

浮動小数点値の双曲線正弦を計算します。

[指定形式]

```
#include <mathf.h>
float sinhf(float f);
```

[引数]

f　双曲線正弦を求める浮動小数点値

[戻り値]

f の双曲線正弦値

tanhf

浮動小数点値の双曲線正接を計算します。

[指定形式]

```
#include <mathf.h>
float tanhf(float f);
```

[引数]

f　双曲線正接を求める浮動小数点値

[戻り値]

f の双曲線正接値

expf

浮動小数点値の指数関数を計算します。

[指定形式]

```
#include <mathf.h>
float expf(float f);
```

[引数]

f　指数関数を求める浮動小数点値

[戻り値]

f の指数関数値

frexpf

浮動小数点値を [0.5,1.0) の値と 2 のべき乗の積に分解します。

[指定形式]

```
#include <mathf.h>
float frexpf(float value, long *exp);
```

[引数]

value　[0.5,1.0) の値と 2 のべき乗の積に分解する浮動小数点値
exp　2 のべき乗値を格納する記憶域へのポインタ

[戻り値]

value が 0.0 の時：0.0
value が 0.0 でない時：ret*$2^{exp\ の指している領域の値}$ =value で定義される ret の値

[備考]

frexpf 関数は、value を [0.5,1.0) の値と 2 のべき乗の積に分解します。exp の指す領域には、分解した結果の 2 のべき乗値を設定します。
リターン値 ret の値の範囲は [0.5,1.0) または 0.0 になります。
value が 0.0 ならば、exp の指す int 型の記憶域の内容と ret の値は 0.0 になります。

ldexpf

浮動小数点値と 2 のべき乗の積を計算します。

[指定形式]

```
#include <mathf.h>
float ldexpf(float e, long f);
```

[引数]

e　2 のべき乗との積を求める浮動小数点値
f　2 のべき乗値

[戻り値]

$e*2^f$ の演算結果の値

logf

浮動小数点値の自然対数を計算します。

[指定形式]

```
#include <mathf.h>
float logf(float f);
```

[引数]

f　自然対数を求める浮動小数点値

[戻り値]

正常：f の自然対数の値
異常：定義域エラーの時は、非数を返します

[備考]

f の値が負の時、定義域エラーになります。
f の値が 0.0 の時、範囲エラーになります。

log10f

浮動小数点値の 10 を底とする対数を計算します。

[指定形式]

```
#include <mathf.h>
float log10f (float f);
```

[引数]

f　10 を底とする対数を求める浮動小数点値

[戻り値]

正常：f は 10 を底とする対数値
異常：定義域エラーの時は、非数を返します

[備考]

f の値が負の値の時、定義域エラーになります。
f の値が 0.0 の時、範囲エラーになります。

modff

浮動小数点値を整数部分と小数部分に分解します。

[指定形式]

```
#include <mathf.h>
float modff(float a, float *b);
```

[引数]

a 整数部分と小数部分に分解する浮動小数点値
b 整数部分を格納する記憶域を指すポインタ

[戻り値]

a の小数部分

powf

浮動小数点値のべき乗を計算します。

[指定形式]

```
#include <mathf.h>
float powf(float x, float y);
```

[引数]

x　べき乗される値
y　べき乗する値

[戻り値]

正常：x の y 乗の値
異常：定義域エラーの時は、非数を返します

[備考]

x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。

sqrtf

浮動小数点値の正の平方根を計算します。

[指定形式]

```
#include <mathf.h>
float sqrtf(float f);
```

[引数]

f　正の平方根を求める浮動小数点値

[戻り値]

正常：f の正の平方根の値
異常：定義域エラーの時は、非数を返します

[備考]

f の値が負の値の時、定義域エラーになります。

ceilf

浮動小数点値の小数点以下を切り上げた整数値を求めます。

[指定形式]

```
#include <mathf.h>
float ceilf(float f);
```

[引数]

f　小数点以下を切り上げる浮動小数点値

[戻り値]

f の小数点以下を切り上げた整数値

[備考]

ceilf 関数は、f の値より大きい、または等しい最小の整数値を float 型の値として返す関数です。したがって f の値が負の値の時は小数点以下を切り捨てた時の値を返します。

fabsf

浮動小数点値の絶対値を計算します。

[指定形式]

```
#include <mathf.h>
float fabsf(float f);
```

[引数]

f　絶対値を求める浮動小数点値

[戻り値]

f の絶対値

floorf

浮動小数点値の小数点以下を切り捨てた整数値を求めます。

[指定形式]

```
#include <mathf.h>
float floorf(float f);
```

[引数]

f　小数点以下を切り捨てる浮動小数点値

[戻り値]

f の小数点以下を切り捨てた整数値

[備考]

floorf 関数は、f の値を超えない範囲の整数の最大値を、float 型の値として返す関数です。したがって f の値が負の値の時は小数点以下を切り上げた時の値を返します。

fmodf

浮動小数点値どうしを除算した結果の余りを計算します。

[指定形式]

```
#include <mathf.h>
float fmodf(float x, float y);
```

[引数]

x 被除数
y 除数

[戻り値]

y の値が 0.0 の時：x
y の値が 0.0 でない時：x を y で除算した結果の余り

[備考]

fmodf 関数では、引数 x、y、リターン値 ret の間には、次に示す関係が成立します。
x=y*i+ret（ただし i は整数値）
また、リターン値 ret の符号は x の符号と同じ符号になります。
x/y の商を表現できない場合、結果の値は、保証しません。

7.4.9 <setjmp.h>

関数間の制御の移動をサポートします。
以下のマクロは、処理系定義です。

種別	定義名	説明
型 (マクロ)	jmp_buf	関数間の制御の移動を可能とする情報を保存しておくための記憶域に対応する型名です。
関数	setjmp	現在実行中の関数の jmp_buf で定義した実行環境を指定した記憶域に退避します。
	longjmp	setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。

setjmp 関数は現在の関数の実行環境を退避します。その後 longjmp 関数を呼び出すことにより、setjmp 関数を呼び出したプログラム上の位置に戻ることができます。

以下に setjmp、longjmp 関数を使用して関数間の制御の移動をサポートした例を示します。

```
1   #include <stdio.h>
2   #include <setjmp.h>
3   jmp_buf env;
4   void sub();
5   void main()
6   {
7
8       if (setjmp(env)!=0){
9           printf("return from longjmp\n");
10          exit(0);
11      }
12      sub();
13  }
14
15  void sub()
16  {
17      printf("subroutine is running \n");
18      longjmp(env,1);
19  }
```

【説明】
8 行目で setjmp 関数を呼んでいます。この時、setjmp 関数の呼び出された環境を、jmp_buf 型の変数 env に退避します。この時のリターン値は 0 なので、次に関数 sub が呼び出されます。
関数 sub の中で呼び出される longjmp 関数により、変数 env に退避した環境を回復します。その結果、プログラムは、あたかも 8 行目の setjmp 関数からリターンしたかのようにふるまいます。ただし、この時のリターン値は longjmp 関数の第 2 実引数で指定した値 (1) になります。
その結果、9 行目以降が実行されます。

setjmp

現在実行中の関数の実行環境を、指定した記憶域に退避します。

[指定形式]

```
#include <setjmp.h>
long setjmp(jmp_buf env);
```

[引数]

env　実行環境を退避する記憶域へのポインタ

[戻り値]

setjmp 関数を呼び出した時：0
longjmp 関数からのリターン時：0 以外

[備考]

setjmp 関数により退避された実行環境は、longjmp 関数において使用されます。
setjmp 関数として呼び出された時のリターン値は 0 ですが、longjmp 関数からリターンしてきた時のリターン値は、longjmp 関数で指定した第 2 引数の値となります。
setjmp 関数を複雑な式から呼び出す場合、式の評価の途中結果等の現在の実行環境の一部が失われる可能性があります。setjmp 関数は setjmp 関数の結果と定数式の比較という形だけで使用し、複雑な式の中では呼び出さないようにしてください。
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

setjmp 関数で退避していた関数の実行環境を回復し、setjmp 関数を呼び出したプログラムの位置に制御を移動します。

[指定形式]

```
#include <setjmp.h>
void longjmp(jmp_buf env, long ret);
```

[引数]

env　実行環境を退避した記憶域へのポインタ
ret　setjmp 関数へのリターンコード

[備考]

longjmp 関数は、同じプログラム中で最後に呼び出された setjmp 関数によって退避された関数の実行環境を第 1 引数 env で指定された記憶域から回復し、その setjmp 関数を呼び出したプログラムの位置に制御を移します。この時 longjmp 関数の第 2 引数 ret が setjmp 関数のリターン値として返ります。ただし、ret が 0 の時は setjmp 関数へのリターン値としては 1 が返ります。

setjmp 関数が呼び出されていない時、あるいは setjmp 関数を呼び出した関数がすでに return 文を実行している時は、longjmp 関数の動作は保証しません。

7.4.10 <stdarg.h>

可変個の引数を持つ関数に対し、その引数の参照を可能にします。

以下はすべて処理系定義です。

種別	定義名	説明
型 (マクロ)	va_list	可変個の引数を参照するために、va_start, va_arg, va_end マクロで共通に使用される変数の型です。
関数 (マクロ)	va_start	可変個の引数の参照を行うため、初期設定処理を行います。
	va_arg	可変個の引数を持つ関数に対して、現在参照中引数の次の引数への参照を可能とします。
	va_end	可変個の引数を持つ関数の引数への参照を終了させます。
	va_copy <-*lang=c99*>	可変個の引数をコピーします。

本標準インクルードファイルで定義しているマクロを使用したプログラムの例を以下に示します。

```
1   #include <stdio.h>
2   #include <stdarg.h>
3
4   extern void prlist(int count,...);
5
6   void main()
7   {
8       prlist(1, 1);
9       prlist(3, 4, 5, 6);
10      prlist(5, 1, 2, 3, 4, 5);
11  }
12
13  void prlist(int count,...)
14  {
15      va_list ap;
16      int i;
17
18      va_start(ap,count);
19      for(i=0;i<count;i++)
20          printf("%d",va_arg(ap,int));
21      putchar('\n');
22      va_end(ap);
23  }
```

【説明】

この例では、第 1 引数に出力するデータの数を指定し、以下の引数をその数だけ出力する関数 prlist を実現しています。

18 行目で、可変個の引数への参照を va_start で初期化します。その後引数を一つ出力するたびに、va_arg マクロによって次の引数を参照します (20 行目)。va_arg マクロでは、引数の型名 (この場合は int 型) を第 2 引数に指定します。

引数の参照が終了したら、va_end マクロを呼び出します (22 行目)。

va_start

可変個の実引数への参照を行うため、初期設定処理を行います。

[指定形式]

```
#include <stdarg.h>
void va_start(va_list ap, parmN);
```

[引数]

ap　可変個の引数にアクセスするための変数
parmN　最右端の引数の識別子

[備考]

va_start マクロは、va_arg、va_end マクロによって使用される ap の初期化を行います。また、parmN には、外部関数定義における引数の並びの最右端の引数の識別子、すなわち「,...」の直前の識別子を指定します。
可変個の名前のない引数を参照するためには、va_start マクロ呼び出しを一番初めに実行する必要があります。

va_arg

可変個の実引数を持つ関数に対して、現在参照中の引数の次の引数への参照を可能とします。

[指定形式]

```
#include <stdarg.h>
type va_arg(va_list ap, type);
```

[引数]

ap　可変個の引数にアクセスするための変数
type　アクセスする引数の型

[戻り値]

引数の値

[備考]

va_start マクロで初期化した va_list 型の変数を第 1 引数に指定します。
ap の値は va_arg を使用する度に更新され、結果として可変個の引数が順次本マクロのリターン値として返されます。
第 2 引数 type は、参照する型を指定してください。
ap は va_start によって初期化された ap と同じでなければなりません。
type に char 型、unsigned char 型、short 型、unsigned short 型、float 型のように関数の引数に指定した時に型変換によってサイズが変わる型を指定した場合、正しく引数を参照することができなくなります。このような型を指定すると動作は保証しません。

va_end

可変個の実引数を持つ関数の引数への参照を終了させます。

[指定形式]

```
#include <stdarg.h>
void va_end(va_list ap);
```

[引数]

ap　可変個の引数を参照するための変数

[備考]

ap は va_start によって初期化された ap と同じでなければなりません。また、関数からの return 前に va_end マクロが呼び出されない時は、その関数の動作は保証しません。

va_copy

可変個の実引数を持つ関数に対して、現在参照中の引数の複製を作ります。

[指定形式]

```
#include <stdarg.h>
void va_copy(va_list dest, va_list src);
```

[引数]

dest　可変個の引数を参照するための変数の複製
src　可変個の引数を参照するための変数

[備考]

va_start マクロで初期化され、va_arg で使用された可変個の引数の状態を持つ第 2 引数 src に対し、第 1 引数 dest に複製を作ります。
src は va_start によって初期化された src と同じでなければなりません。
dest は、これ以降の va_arg マクロで可変個の引数を表す引数として使用することができます。

7.4.11 <stdio.h>

ストリーム入出力用ファイルの入出力に関する処理を行います。

以下の定数 (マクロ) はすべて処理系定義です。

種別	定義名	説明
定数 (マクロ)	FILE	ストリーム入出力処理で必要とするバッファへのポインタやエラー指示子、終了指示子などの各種制御情報を保存しておく構造体の型です。
	_IOFBF	バッファ領域の使用方法として、入出力処理はすべてバッファ領域を使用することを示しています。
	_IOLBF	バッファ領域の使用方法として、入出力処理は行単位でバッファ領域を使用することを示しています。
	_IONBF	バッファ領域の使用方法として、入出力処理はバッファ領域を使用しないことを示しています。
	BUFSIZ	入出力処理において必要とするバッファの大きさです。
	EOF	ファイルの終わり (End Of File) すなわちファイルからそれ以上の入力が無いことを示しています。
	L_tmpnam*	tmpnam 関数によって生成される一時ファイル名の文字列を格納するのに十分な大きさの配列のサイズです。
	SEEK_CUR	ファイルの現在の読み書き位置を現在の位置からのオフセットに移すことを示しています。
	SEEK_END	ファイルの現在の読み書き位置をファイルの終了位置からのオフセットに移すことを示しています。
	SEEK_SET	ファイルの現在の読み書き位置をファイルの先頭位置からのオフセットに移すことを示しています。
	SYS_OPEN*	処理系が同時にオープンすることができることを保証するファイルの数です。
	TMP_MAX*	tmpnam 関数によって生成される一意なファイル名の個数の最大値です。
	stderr	標準エラーファイルに対するファイルポインタです。
	stdin	標準入力ファイルに対するファイルポインタです。
	stdout	標準出力ファイルに対するファイルポインタです。

種別	定義名	説明
関数	fclose	ストリーム入出力用ファイルをクローズします。
	fflush	ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。
	fopen	ストリーム入出力用ファイルを指定したファイル名によってオープンします。
	freopen	現在オープンされているストリーム入力出用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。
	setbuf	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
	setvbuf	ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。
	fprintf	書式に従ってストリーム入出力用ファイルへデータを出力します。
	vfprintf	可変個の引数リストを書式に従って指定したストリーム入出力用ファイルに出力します。
	printf	データを書式に従って変換し、標準出力ファイル (stdout) へ出力します。
	vprintf	可変個の引数リストを書式に従って標準出力ファイル (stdout) に出力します。
	sprintf	データを書式に従って変換し、指定した領域へ出力します。
	sscanf	指定した記憶域からデータを入力し、書式に従って変換します。
	snprintf <*-lang=c99*>	データを書式に従って変換し、配列に書き込みます。
	vsnprintf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた snprintf と等価です。
	vfscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた fscanf と等価です。
	vscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた scanf と等価です。
	vsscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた sscanf と等価です。
	fscanf	ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。
	scanf	標準入力ファイル (stdin) からデータを入力し、書式に従って変換します。
	vsprintf	可変個の引数リストを書式に従って指定した領域に出力します。
	fgetc	ストリーム入出力用ファイルから 1 文字入力します。
	fgets	ストリーム入出力用ファイルから文字列を入力します。
	fputc	ストリーム入出力用ファイルへ 1 文字出力します。
	fputs	ストリーム入出力用ファイルへ文字列を出力します。
	getc	(マクロ) ストリーム入出力用ファイルから 1 文字入力します。
	getchar	(マクロ) 標準入力ファイルから 1 文字入力します。
	gets	標準入力ファイルから文字列を入力します。
	putc	(マクロ) ストリーム入出力用ファイルへ 1 文字出力します。
	putchar	(マクロ) 標準出力ファイルへ 1 文字出力します。

種別	定義名	説明
関数	puts	標準出力ファイルへ文字列を出力します。
	ungetc	ストリーム入出力用ファイルへ 1 文字をもどします。
	fread	ストリーム入出力用ファイルから指定した記憶域にデータを入力します。
	fwrite	記憶域からストリーム入出力用ファイルにデータを出力します。
	fseek	ストリーム入出力用ファイルの現在の読み書き位置を移動させます。
	ftell	ストリーム入出力用ファイルの現在の読み書き位置を求めます。
	rewind	ストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
	clearerr	ストリーム入出力用ファイルのエラー状態をクリアします。
	feof	ストリーム入出力用ファイルが終わりであるかどうかを判定します。
	ferror	ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
	perror	標準エラーファイル (stderr) に、エラー番号に対応したエラーメッセージを出力します。
型	fpos_t	ファイル中の任意の位置を指定可能な型です。
定数 (マクロ)	FOPEN_MAX	同時にオープン可能なファイル数です。
	FILENAME_MAX	保持可能なファイル名の最大長です。

注 * 本処理系では、定義されません。

処理系定義仕様

	項目	コンパイラの仕様
1	入力テキストの最終の行が終了を示す改行文字を必要とするかどうか	規定しません。低水準インタフェースルーチンの仕様によります。
2	改行文字の直前に書き出された空白文字は、読み込み時に読み込まれるかどうか	
3	バイナリファイルに書かれたデータに付加されるヌル文字の数	
4	追加モード時のファイル位置指定子の初期値	
5	テキストファイルへの出力によってそれ以降のファイルのデータが失われるかどうか	
6	ファイルバッファリングの仕様	
7	長さ 0 のファイルが存在するかどうか	
8	正当なファイル名の構成規則	
9	同時に同じファイルをオープンできるかどうか	
10	fprintf 関数における %p 書式変換の出力形式	16 進数出力となります
11	fscanf 関数における %p 書式変換の入力形式 fscanf 関数での変換文字「-」の意味	16 進数入力となります。 先頭、最後あるいは「^」の直後でない場合、直前の文字と直後の範囲を示します。

	項目	コンパイラの仕様
12	fgetpos,ftell 関数で設定される errno の値	fgetpos 関数はサポートしていません。 ftell 関数については規定しません。 低水準インタフェースルーチンの仕様によります。
13	perror 関数が生成するメッセージ出力形式	メッセージの出力形式を (a) に示します。

(a) perror 関数の出力形式は、
< 文字列 > : <error に設定したエラー番号に対応するエラーメッセージ >
となります。
(b) printf 関数、fprintf 関数で、浮動小数点の無限大および非数を表示するときの形式を表 7.8 に示します。

表 7.8 無限大および非数の表示形式

	値	表示形式
1	正の無限大	＋＋＋＋＋＋
2	負の無限大	－－－－－－
3	非数	＊＊＊＊＊＊

ストリーム入出力用ファイルに対する一連の入出力処理を行ったプログラムの例を以下に示します。

```
1   #include <stdio.h>
2
3   void main()
4   {
5       int c;
6       FILE *ifp, *ofp;
7
8       if ((ifp=fopen("INPUT.DAT","r"))==NULL){
9           fprintf(stderr,"cannot open input file\n");
10          exit(1);
11      }
12      if ((ofp=fopen("OUTPUT.DAT","w"))==NULL){
13          fprintf(stderr,"cannot open output file\n");
14          exit(1);
15      }
16      while ((c=getc(ifp))!=EOF)
17          putc(c, ofp);
18      fclose(ifp);
19      fclose(ofp);
20  }
```

【説明】
ファイル INPUT.DAT の内容をファイル OUTPUT.DAT へコピーするプログラムです。
8 行目の fopen 関数で入力ファイル INPUT.DAT を、12 行目の fopen 関数で出力ファイル OUTPUT.DAT をオープンします。オープンに失敗した場合、fopen 関数のリターン値として NULL が返されますので、エラーメッセージを出力してプログラムを終了させます。
fopen 関数が正常に終了した時、オープンしたファイルの情報を格納するデータ (FILE 型) へのポインタが返されますので、これらを変数 ifp、ofp に設定します。
オープンが成功した後は、これらの FILE 型のデータを用いて入出力を行います。
ファイルの処理が終了したら、fclose 関数でファイルをクローズします。

fclose

ストリーム入出力用ファイルをクローズします。

[指定形式]

```
#include <stdio.h>
long fclose(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

fclose 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルをクローズします。
fclose 関数は、ストリーム入出力用ファイルの出力ファイルがオープンされており、まだ出力されていないデータがバッファに残っている時は、それをファイルに出力してからクローズします。
また、入出力用のバッファがシステムによって自動的に割り付けられていた場合は、それを解放します。

fflush

ストリーム入出力用ファイルのバッファの内容をファイルへ出力します。

[指定形式]

```
#include  <stdio.h>
long fflush(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

fflush 関数は、ストリーム入出力用ファイルの出力ファイルがオープンされている時、ファイルポインタ fp で指定されたストリーム入出力用ファイルのバッファの未出力内容をファイルに出力します。また、入力ファイルがオープンされている時、ungetc 関数の指定を無効にします。

fopen

ストリーム入出力用ファイルを、指定したファイル名によってオープンします。

[指定形式]

```
#include <stdio.h>
FILE *fopen(const char *fname, const char *mode);
```

[引数]

fname　ファイル名を示す文字列へのポインタ
mode　ファイルアクセスモードを示す文字列へのポインタ

[戻り値]

正常：オープンしたファイルのファイル情報を指すファイルポインタ
異常：NULL

[備考]

fopen 関数は、fname が指す文字列をファイル名とするストリーム入出力用ファイルをオープンします。書き出しモードあるいは追加モードで存在しないファイルをオープンしようとした時は、可能な限り新しいファイルを作成します。また既存のファイルに対して書き出しモードでオープンした時は、ファイルの先頭から書き込みが行われ、以前に書き込まれていたファイルの内容は消去されます。

追加モードでオープンしたファイルは、そのファイルの終わりの位置から書き出しの処理が行われます。更新モードでオープンしたファイルは、このファイルに対して入力と出力の両方の処理を行うことができます。

ただし、出力処理は後に fflush、fseek、rewind 関数が実行されることなしに入力処理を続けることはできません。

また同様に入力処理の後に fflush、fseek、rewind 関数が実行されることなしに出力処理を続けることはできません。また、ファイルアクセスモードを示す文字列の後にオープンの方法を指示する文字が付くこともあります。

freopen

現在オープンされているストリーム入出力用ファイルをクローズし、新しいファイルを指定したファイル名で再オープンします。

[指定形式]

```
#include <stdio.h>
FILE *freopen(const char *fname, const char *mode, FILE *fp);
```

[引数]

fname　新しいファイル名を示す文字列へのポインタ
mode　ファイルアクセスモードを示す文字列へのポインタ
fp　現在オープンされているストリーム入出力用ファイルのファイルポインタ

[戻り値]

正常：fp
異常：NULL

[備考]

freopen 関数は、まず、ファイルポインタ fp の示すストリーム入出力用ファイルをクローズします (このクローズ処理が正しく行われない時でも以下の処理は続けます)。
次に、その fp の指す FILE 構造体を再使用して、ファイル名 fname で示すファイルを、ストリーム入出力用にオープンします。
freopen 関数は一時にオープンするファイル数が限られているときなどに有効です。
freopen 関数は通常、fp と同じ値を返しますが、エラーが発生した時は、NULL を返します。

setbuf

ストリーム入出力用のバッファ領域をユーザプログラム側で定義して設定します。

[指定形式]

```
#include <stdio.h>
void setbuf(FILE *fp, char buf[BUFSIZ]);
```

[引数]

fp　ファイルポインタ
buf　バッファ領域へのポインタ

[備考]

setbuf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対して buf の指す記憶域を入出力用のバッファ領域として使用するように定義します。この結果、大きさが BUFSIZ のバッファ領域を使用した入出力処理が行われます。

setvbuf

ストリーム入出力用のバッファ領域をユーザプログラムの側で定義して設定します。

[指定形式]

```
#include <stdio.h>
long setvbuf(FILE *fp, char *buf, long type, size_t size);
```

[引数]

fp　ファイルポインタ
buf　バッファ領域へのポインタ
type　バッファの管理方式
size　バッファ領域の大きさ

[戻り値]

正常：0
異常：0 以外

[備考]

setvbuf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対して buf の指す記憶域を入出力用のバッファ領域として使用するように定義します。
このバッファ領域の使用方法としては、以下の三通りの方法があります。

(a) type に _IOFBF を指定した時
入出力処理はすべてバッファ領域を使用して行います。

(b) type に _IOLBF を指定した時
入出力処理は行単位でバッファ領域を使用して行います。すなわち、入出力データは、改行文字が書かれた時、バッファ領域が一杯になった時、入力が要求された時にバッファ領域から取り出されることになります。

(c) type に _IONBF を指定した時
入出力処理はバッファ領域を使用せず行います。
setvbuf 関数は通常 0 を返しますが、type あるいは size に不正な値が与えられた時、あるいはバッファ領域の使用方法等の要求が受け入れられなかった時には 0 以外の値を返します。
バッファ領域は、オープンされているストリーム入出力用ファイルがクローズされる前に解放してはいけません。また、setvbuf 関数は、ストリーム入出力用ファイルがオープンされてから入出力用処理が行われるまでの間で使用してください。

fprintf

書式に従って、ストリーム入出力用ファイルへデータを出力します。

[指定形式]

```
#include <stdio.h>
long fprintf(FILE *fp, const char *control [, arg] ...);
```

[引数]

fp　ファイルポインタ
control　書式を示す文字列へのポインタ
arg,...　書式に従って出力されるデータの並び

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

fprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、ファイルポインタ fp の示すストリーム入出力用ファイルへ出力します。
fprintf 関数は、通常は変換し出力したデータの個数を返しますが、エラー発生時には負の値を返します。
書式の仕様は以下のとおりです。

【書式の概要】
書式を表す文字列は、2 種類の文字列から構成されます。

- 通常の文字
  次の変換仕様を示す文字列以外の文字はそのまま出力されます。
- 変換仕様
  変換仕様は、% で始まる文字列で、後に続く引数の変換方法を指定します。変換仕様の形式は次の規則に従います。

%[フラグ・・・] { [＊] / [フィールド幅] } (. { [＊] / [精度] }) [パラメータのサイズ指定] 変換文字

この変換仕様に対して、実際に出力する引数が無い時は、その動作は保証しません。また、変換仕様よりも実際に出力する引数の個数が多い時は、余分な引数はすべて無視されます。

【変換仕様の説明】

(a)　フラグ
符号を付けるなどの出力するデータに対する修飾を指定します。指定できるフラグの種類と意味を表 7.9 に示します。

表 7.9　フラグの種類と意味

	種類	意味
1	-	変換したデータの文字数が指定したフィールド幅より少ない時、そのデータをフィールド内で左詰めにして出力します。
2	+	符号付きのデータに変換する時、そのデータの符号に従って、変換したデータの先頭にプラスあるいはマイナス符号を付けます。

	種類	意味
3	空白	符号付きのデータの変換において、変換したデータの先頭に符号が付かない時、そのデータの先頭に空白を付けます。 「＋」と共に使用した時、本フラグは無視されます。
4	#	表 7.11 で説明する変換の種類に従って、変換後のデータに修飾を行います。 1. c、d、i、s、u 変換の時 本フラグは無視されます。 2. o 変換の時 変換したデータの先頭に 0 を付けます。 3. x(あるいは X) 変換の時 変換したデータの先頭に 0x(あるいは 0X) を付けます。 4. e、E、f、g、G 変換の時 変換したデータに小数点以下がない時でも、小数点を出力します。 また、g、G 変換の時は、変換後のデータの後に付く 0 は取り除きません。

(b) フィールド幅

変換したデータを出力する文字数を任意の 10 進数で指定します。
変換したデータの文字数がフィールド幅より少ない時、フィールド幅までそのデータの先頭に空白が付けられます (ただし、フラグとして '-' を指定した時は、データの後に空白が付けられます)。
もし、変換したデータの文字数がフィールド幅より大きい時は、フィールド幅は、変換結果を出力できる幅に拡張されます。
また、フィールド幅指定の先頭が 0 で始まっている時は、出力するデータの先頭には空白ではなく文字「0」が付けられます。

(c) 精度

表 7.11 で説明する変換の種類に従って変換したデータの精度を指定します。
精度は、ピリオド (.) の後に 10 進整数を続ける形式で指定します。10 進整数を省略した時は、0 を指定したものと仮定します。
精度を指定した結果、フィールド幅の指定との間に矛盾が生じれば、フィールド幅の指定を無効とします。
各変換の種類と精度指定の意味を以下に示します。

- d、i、o、u、x、X 変換の時
  変換したデータの最小の桁数を示します。
- e、E、f 変換の時
  変換したデータの小数点以下の桁数を示します。
- g、G 変換の時
  変換したデータの最大有効桁数を示します。
- s 変換の時
  印字される最大文字数を示します。

(d) パラメータのサイズ指定

d、i、o、u、x、X、e、E、f、g、G 変換の時 (表 7.10 参照)
変換するデータのサイズ (short 型、long 型、long long 型、long double 型) を指定します。これ以外の変換の時は、本指定を無視します。表 7.10 にサイズ指定の種類とその意味を示します。

表 7.10 パラメータのサイズ指定の種類とその意味

	種類	意味
1	hh	d、i、o、u、x、X、a、A、e、E、f、F、g および G 変換において、変換するデータが signed char 型あるいは unsigned char 型であることを指定します。n 変換において、変換するデータが signed char 型へのポインタ型であることを指定します。
2	h	d、i、o、u、x、X 変換において、変換するデータが short 型あるいは unsigned short 型であることを指定します。n 変換において、変換するデータが short 型へのポインタ型であることを指定します。
3	l	d、i、o、u、x、X 変換において、変換するデータが long 型、unsigned long 型あるいは、double 型であることを指定します。
4	L	e、E、f、g、G 変換において、変換するデータが long double 型であることを指定します。

	種類	意味
5	ll	d、i、o、u、x、X 変換において、変換するデータが long long 型あるいは、unsigned long long 型であることを指定します。n 変換において、変換するデータが long long 型へのポインタ型であることを指定します。
6	j	d、i、o、u、x、X 変換において、変換するデータが intmax_t 型あるいは、uintmax_t 型であることを指定します。n 変換において、変換するデータが size_t 型へのポインタ型であることを指定します。
7	z	d、i、o、u、x、X 変換において、変換するデータが size_t 型あるいは、size_t 型に対応する符号付き整数型であることを指定します。n 変換において、変換するデータが size_t 型へのポインタ型であることを指定します。
8	t	d、i、o、u、x、X 変換において、変換するデータが ptrdiff_t 型あるいは、ptrdiff_t 型に対応する符号なし整数型であることを指定します。n 変換において、変換するデータが ptrdiff_t 型へのポインタ型であることを指定します。

(e) 変換文字

変換するデータをどのような形式に変換するかを指定します。
もし、変換するデータが構造体や配列型の時や、それらの型を指すポインタの時は、s 変換で文字の配列を変換する時、p 変換でポインタを変換する時を除いてその動作は保証しません。
表 7.11 に変換文字と変換方式を示します。この表に述べられていない英文字を変換文字として指定した時は、その動作は保証しません。また、それ以外の文字を指定した時の動作はコンパイラによって異なります。

表 7.11 変換文字と変換の方式

	変換文字	変換の種類	変換の方式	変換対象の型	精度に関する注意事項
1	d	d 変換	int型データを符号付き10進数の文字列に変換します。d 変換と i 変換は同じ仕様です。	int 型	精度指定は、最低で何文字出力されるかを示しています。もし、変換後の文字数が精度の値より少ない時は、文字列の先頭に 0 が付きます。また、精度を省略した時は、1 が仮定されます。さらに、0 の値を持つデータを精度に 0 を指定して変換し出力しようとした時は、何も出力されません。
2	i	i 変換		int 型	
3	o	o 変換	int 型データを符号なしの 8 進数の文字列に変換します。	int 型	
4	u	u 変換	int型データを符号なしの10進数の文字列に変換します。	int 型	
5	x	x 変換	int型データを符号なしの16進数に変換します。16 進文字には a、b、c、d、e、f を用います。	int 型	
6	X	X 変換	int型データを符号なしの16進数に変換します。16 進文字には A、B、C、D、E、F を用います。	int 型	
7	f	f 変換	double 型データを「[-]ddd.ddd」の形式の 10 進数の文字列に変換します。	double 型	精度の指定は、小数点以降の桁数を表します。小数点以降の文字が存在する時には、必ず小数点の前に 1 桁の数字が出力されます。精度を省略した時は、6 が仮定されます。また、精度に 0 を指定した時は、小数点と小数点以降の文字は出力しません。出力するデータは丸められます。
8	F	F 変換		double 型	

	変換文字	変換の種類	変換の方式	変換対象の型	精度に関する注意事項
9	e	e 変換	double 型データを「[-]d.ddde±dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	精度の指定は、小数点以降の桁数を表します。変換した文字は小数点の前に 1 桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される形式となります。精度を省略した時は 6 が仮定されます。また、精度に 0 を指定した時は、小数点以降の文字は出力しません。出力するデータは丸められます。
10	E	E 変換	double 型データを「[-]d.dddE±dd」の形式の 10 進数の文字列に変換します。指数は、少なくとも 2 桁出力されます。	double 型	
11	g	g変換(あるいは G 変換)	変換する値と有効桁数を指定する精度の値から f 変換の形式で出力するか e 変換 (あるいは E 変換) の形式で出力するかを決め double 型データを出力します。もし、変換されたデータの指数が -4 より小さいか、有効桁数を指定する精度より大きい時には e 変換 (あるいは E 変換) の形式に変換します。	double 型	精度の指定は、変換されたデータの最大有効桁数を示します。
12	G			double 型	
13	a	a 変換	double 型データを「[-]0xh.hhhhp±d」の形式の 16 進数の文字列に変換します。指数は、少なくとも 1 桁出力されます。	double 型	精度の指定は、小数点以降の桁数を表します。変換した文字は小数点の前に 1 桁の数字が出力され、小数点以降に精度に等しい桁数の数字が出力される形式となります。精度を省略した時は、正確な値を表現するのに充分な精度が仮定されます。また、精度に 0 を指定した時は、小数点以降の文字は出力しません。出力するデータは丸められます。
14	A	A 変換	double 型データを「[-]0Xh.hhhhP±d」の形式の 16 進数の文字列に変換します。指数は、少なくとも 1 桁出力されます。	double 型	
15	c	c 変換	int 型のデータを unsigned char 型データとし、そのデータに対応する文字に変換します。	int 型	精度の指定は無効です。
16	s	s 変換	char 型へのポインタ型データが指す文字列を文字列の終了を示すヌル文字まで、あるいは、精度で指定された文字数分出力します (ただしヌル文字は出力されません。また、空白、水平タブ、改行文字は変換文字列に含まれません)。	char型へのポインタ型	精度の指定は出力する文字数を示します。もし、精度が省略された時は、データが指す文字列のヌル文字までの文字が出力されます (ただし、ヌル文字は出力されません。また、空白、水平タブ、改行文字は変換文字列に含まれません)。
17	p	p 変換	データをポインタとして、コンパイラごとに定義された印字可能な文字列に変換します。	void 型へのポインタ	精度の指定は無効です。

	変換文字	変換の種類	変換の方式	変換対象の型	精度に関する注意事項
18	n	データの変換は生じません。	データは int 型へのポインタ型とみなされ、このデータが指す記憶域にいままで、出力したデータの文字数を設定します。	int 型へのポインタ型	
19	%	データの変換は生じません。	% を出力します。	なし	

(f) フィールド幅あるいは精度に対する * 指定

フィールド幅あるいは精度指定の値として * を指定することができます。この時は、この変換仕様に対応するパラメータの値がフィールド幅あるいは精度指定の値として使用されます。このパラメータが負のフィールド幅を持つ時は、正のフィールド幅にフラグ - が指定されたと解釈します。また、負の精度を持つ時は、精度が省略されたものと解釈します。

vfprintf

可変個の引数リストを書式に従って、指定したストリーム入出力用ファイルに出力します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vfprintf(FILE *fp, const char *control, va_list arg);
```

[引数]

fp　ファイルポインタ
control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

vfprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、fp の示すストリーム入出力用ファイルへ出力します。
vfprintf 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。
また、vfprintf 関数では va_end マクロは呼び出しません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ) によって初期化されていなければなりません。

printf

データを書式に従って変換し、標準出力ファイル (stdout) へ出力します。

[指定形式]

```
#include <stdio.h>
long printf(const char *control[, arg] ...);
```

[引数]

control　書式を示す文字列へのポインタ
arg,...　書式に従って出力されるデータ

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

printf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、標準出力ファイル (stdout) へ出力します。
書式の仕様の詳細は fprintf 関数を参照してください。

vprintf

可変個の引数リストを書式に従って標準出力ファイル (stdout) に出力します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vprintf(const char *control, va_list arg);
```

[引数]

control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

vprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、標準出力ファイルへ出力します。
vprintf 関数は、変換し出力したデータの個数を返しますが出力エラーが発生した時は負の値を返します。
また、vprintf 関数では va_end マクロは呼び出しません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ) によって初期化されていなければなりません。

sprintf

データを書式に従って変換し、指定した領域へ出力します。

[指定形式]

```
#include <stdio.h>
long sprintf(char *s, const char *control [, arg] ...);
```

[引数]

s　データを出力する記憶域へのポインタ
control　書式を示す文字列へのポインタ
arg,...　書式に従って出力されるデータ

[戻り値]

変換した文字数

[備考]

sprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、s の指す記憶域へ出力します。
変換して出力した文字列の最後には、ヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。
書式の仕様の詳細は fprintf 関数を参照してください。

sscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdio.h>
long sscanf(const char *s, const char *control [, ptr] ...);
```

[引数]

s　入力するデータがある記憶域
control　書式を示す文字列へのポインタ
ptr,...　入力変換したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：EOF

[備考]

sscanf 関数は、s の指す記憶域からデータを入力し、control が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を ptr の指す記憶域へ格納します。
sscanf 関数は、入力変換に成功したデータの個数を返します。また、最初の変換の前に入力するデータが終了した時には EOF を返します。
書式の仕様の詳細は fscanf 関数を参照してください。

snprintf

データを書式に従って変換し、指定した領域へ出力します。

[指定形式]

```
#include <stdio.h>
long snprintf(char *restrict s, size_t n, const char *restrict control [, arg] ...);
```

[引数]

s　データを出力する記憶域へのポインタ
n　出力する文字数
control　書式を示す文字列へのポインタ
arg,...　書式に従って出力されるデータ

[戻り値]

変換した文字数

[備考]

snprintf 関数は、control が指す書式を示す文字列に従って、引数 arg を変換、編集し、s の指す記憶域へ出力します。
変換して出力した文字列の最後には、ヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。書式の仕様の詳細は fprintf 関数を参照してください。

vsnprintf

データを書式に従って変換し、指定した領域へ出力します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vsnprintf(char *restrict s, size_t n, const char *restrict control, va_list arg);
```

[引数]

s　データを出力する記憶域へのポインタ
n　出力する文字数
control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

変換した文字数

[備考]

vsnprintf 関数は、可変個引数を arg で置き換えた snprintf と等価です。
vsnprintf 関数の呼出し前に、va_start マクロで arg を初期化してください。
vsnprintf 関数は、va_end マクロを呼び出しません。

vfscanf

ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vfscanf(FILE *restrict fp, const char *restrict control, va_list arg);
```

[引数]

fp　ファイルポインタ
control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

vfscanf 関数は可変個引数並びを arg で置き換えた fscanf と等価です。
vfscanf 関数の呼出し前に、va_start マクロで arg を初期化してください。
vfscanf 関数は va_end マクロを呼び出しません。

vscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vscanf(const char *restrict control, va_list arg);
```

[引数]

control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

vscanf 関数は、可変個数引数を arg で置き換えた scanf と等価です。
vscanf 関数の呼出し前に、va_start マクロで arg を初期化してください。
vscanf 関数は va_end マクロを呼びません。

vsscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vsscanf(const char *restrict s, const char *restrict control, va_list arg);
```

[引数]

s　入力するデータがある記憶域
control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

vsscanf 関数は、可変個数引数を arg で置き換えた sscanf と等価です。
vsscanf 関数の呼出し前に、va_start マクロで arg を初期化してください。
vsscanf 関数は va_end マクロを呼びません。

fscanf

ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdio.h>
long fscanf(FILE *fp, const char *control[, ptr] ...);
```

[引数]

fp　ファイルポインタ
control　書式を示す文字列へのポインタ
ptr,...　入力したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

fscanf 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルからデータを入力し、control が指す書式を文字列に従って変換、編集して、その結果を ptr の指す記憶域へ格納します。
データを入力するための書式の仕様を以下に示します。

【書式の概要】
書式を表す文字列は、以下の 3 種類の文字列から構成されます。

- 空白文字
  空白 (' ')、水平タブ ('\t') あるいは改行文字 ('\n') を指定すると、入力データを次の空白類文字でない文字まで読み飛ばす処理を行います。
- 通常の文字
  上の空白文字でも % でもない文字を指定すると、入力データを 1 文字入力します。ここで入力した文字は書式を表す文字列の中に指定した文字と一致していなければなりません。
- 変換仕様
  変換仕様は、% で始まる文字列で、書式を表す文字列の後に続く引数の指す領域に入力データを変換して格納する方法を指定します。変換仕様の形式は次の規則に従います。
  %[*][フィールド幅][変換後のデータのサイズ] 変換文字
  書式中の変換仕様に対して入力したデータを格納する記憶域へのポインタがない時は、その動作は保証しません。また、書式が終了したにもかかわらず、入力データを格納する記憶域へのポインタが残っている時は、そのポインタは無視されます。

【変換仕様の説明】
* 指定
入力したデータを引数が指す記憶域に格納することを抑止します。
フィールド幅
入力するデータの最大文字数を 10 進数字で指定します。
変換後のデータのサイズ
d、i、o、u、x、X、e、E、f 変換の時 (表 7.13 参照)、変換後のデータのサイズ (short 型、long 型、long long 型、long double 型) を指定します。これ以外の変換の時は、本指定を無視します。表 7.12 にサイズ指定の種類とその意味を示します。

表 7.12 変換後のデータのサイズ指定の種類とその意味

	種類	意味
1	h	d、i、o、u、x、X 変換において、変換後のデータは short 型であることを指定します。
2	l	d、i、o、u、x、X 変換において、変換後のデータは long 型であることを指定します。 また、e、E、f 変換において、変換後のデータは double 型であることを指定します。
3	L	e、E、f 変換において、変換後のデータは、long double 型であることを指定します。
4	ll	d、i、o、u、x、X 変換において、変換後のデータは long long 型であることを指定します。

- 変換文字
入力するデータは、各変換文字が指定する変換の種類に従って変換します。ただし、空白類文字を読み込んだ場合、変換の対象として許されていない文字を読み込んだ場合、あるいは指定されたフィールド幅を超えた場合は処理を終了します。

表 7.13 変換文字と変換の内容

	変換文字	変換の種類	変換の方式	対応するパラメータの型名
1	d	d 変換	10 進数字の文字列を整数型データに変換します。	整数型
2	i	i 変換	先頭に符号が付いている 10 進数字の文字列、あるいは最後に u(U)または l(L)が付いている 10 進数字の文字列を整数型データに変換します。また、先頭が 0x(あるいは 0X) で始まっている文字列は、16 進数字として解釈し、文字列を int 型データに変換します。さらに、先頭が 0 で始まっている文字列は、8 進数字として解釈し文字列を int 型データに変換します。	整数型
3	o	o 変換	8 進数字の文字列を整数型データに変換します。	整数型
4	u	u 変換	符号なしの 10 進数字の文字列を整数型データに変換します。	整数型
5	x	x 変換	16 進数字の文字列を整数型データに変換します。 x 変換と X 変換に意味の違いはありません。	整数型
6	X	X 変換		
7	s	s 変換	空白、水平タブ、改行文字を読み込むまでをひとつの文字列として変換します。文字列の最後にはヌル文字を付加します (変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です)。	文字型
8	c	c 変換	1 文字を入力します。この時、入力する文字が空白類文字であっても読み飛ばすことはしません。もし、空白類文字以外の文字だけを読み込む時は、%1s と指定してください。また、フィールド幅が指定されている時は、その指定分の文字が読み込まれます。したがって、この時、変換したデータを格納する記憶域は、指定分の大きさが必要です。	char 型
9	e	e 変換	浮動小数点型を示す文字列を浮動小数点型データに変換します。e 変換と E 変換、g 変換と G 変換にそれぞれ意味の違いはありません。入力形式は strtod 関数で表現できる浮動小数点型です。	浮動小数点型
10	E	E 変換		
11	f	f 変換		
12	g	g 変換		
13	G	G 変換		
14	p	p 変換	fprintf 関数において、p 変換で変換される形式の文字列をポインタ型データに変換します。	void 型へのポインタ型
15	n	データの変換は生じません。	データの入力は行わず、いままでに入力したデータの文字数が設定されます。	整数型

	変換文字	変換の種類	変換の方式	対応するパラメータの型名
16	[	[変換	[の後に文字の集合、その後に] を指定します。この文字集合は、文字列を構成する文字の集合を定義しています。もし、文字集合の最初の文字が ^ でない時は、入力データはこの文字集合にない文字が最初に読み込まれるまでをひとつの文字列として入力します。もし、最初の文字が ^ の時は、^ を除いた文字集合の文字が最初に読み込まれるまでをひとつの文字列として入力します。入力した文字列の最後には自動的にヌル文字を付加します (変換したデータを設定する文字列は、ヌル文字を含めて格納できるサイズが必要です)。	文字型
17	%	データの変換は生じません。	% を読み込みます。	なし

変換文字が表 7.13 に示す文字以外の英文字の時は、その動作は保証しません。また、その他の文字の時は、その動作は処理系定義です。

scanf

標準入力ファイル (stdin) からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdio.h>
long scanf(const char *control[, ptr] ...);
```

[引数]

control　書式を示す文字列へのポインタ
ptr,...　入力変換したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：EOF

[備考]

scanf 関数は、標準入力ファイル (stdin) からデータを入力し、control が指す書式を示す文字列に従って、そのデータを変換、編集して、その結果を ptr の指す記憶域へ格納します。
scanf 関数は、入力変換に成功したデータの個数をリターン値として返します。最初の変換の前に標準入力ファイルが終了した時には EOF を返します。
書式の仕様の詳細は fscanf 関数を参照してください。
%e 変換では、double 型の場合は l、long double 型の場合は L で指定します。デフォルトの型は float 型です。

vsprintf

可変個の引数リストを書式に従って、指定した記憶域に出力します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
long vsprintf(char *s, const char *control, va_list arg);
```

[引数]

s　データを出力する記憶域へのポインタ
control　書式を示す文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換した文字数
異常：負の数

[備考]

vsprintf 関数は、control が指す書式を示す文字列に従って、可変個の引数リストを順に変換、編集し、s により指される記憶域へ出力します。
変換して出力した文字列の最後にヌル文字が付加されます。このヌル文字はリターン値である出力した文字数の中には含まれません。
書式の仕様の詳細は fprintf 関数を参照してください。
引数リストを示す arg は、va_start(およびそれに続く va_arg マクロ) によって初期化されていなければなりません。

fgetc

ストリーム入出力用ファイルから 1 文字入力します。

[指定形式]

```
#include <stdio.h>
long fgetc(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：EOF
　　ファイルの終了でない時：入力した文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してのエラー指示子が設定されます。
fgetc 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから 1 文字入力します。
fgetc 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の時は、EOF を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

fgets

ストリーム入出力用ファイルから文字列を入力します。

[指定形式]

```
#include <stdio.h>
char *fgets(char *s, long n, FILE *fp);
```

[引数]

s　文字列を入力する記憶域へのポインタ
n　文字列を入力する記憶域のバイト数
fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：NULL
　　ファイルの終了でない時：s
異常：NULL

[備考]

fgets 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから、ポインタ s の指す記憶域に文字列を入力します。
fgets 関数は、n-1 文字まであるいは改行文字を入力するまで、またはファイルの終わりになるまで文字を入力し、入力文字列の最後にヌル文字を付け加えます。
fgets 関数は通常、文字列を入力する記憶域へのポインタ s を返しますが、ファイルが終了した時やエラー発生の時は NULL を返します。
ファイルが終了した時は、s が指す記憶域の内容は変化しませんが、エラー発生の時は、s が指す記憶域の内容は保証しません。

fputc

ストリーム入出力用ファイルへ 1 文字出力します。

[指定形式]

```
#include <stdio.h>
long fputc(long c, FILE *fp);
```

[引数]

c　出力する文字
fp　ファイルポインタ

[戻り値]

正常：出力した文字
異常：EOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
fputc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力ファイルへ出力します。
fputc 関数は、通常出力した文字 c を返しますが、エラー発生の時は、EOF を返します。

fputs

ストリーム入出力用ファイルへ文字列を出力します。

[指定形式]

```
#include <stdio.h>
long fputs(const char *s, FILE *fp);
```

[引数]

s　出力する文字列へのポインタ
fp　ファイルポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

fputs 関数は、s の指すヌル文字の直前までの文字列をファイルポインタ fp の示すストリーム入出力用ファイルへ出力します。この時、文字列の終了を示すヌル文字は出力されません。
fputs 関数は、通常 0 を返しますが、エラー発生の時は、0 以外の値を返します。

getc

ストリーム入出力用ファイルから 1 文字入力します。

[指定形式]

```
#include <stdio.h>
long getc(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：EOF
　　ファイルの終了でない時：入力した文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。
getc 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから 1 文字入力します。
getc 関数は、通常入力した 1 文字を返しますがファイルの終了やエラー発生の時は、EOF を返します。またファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

getchar

標準入力ファイル (stdin) から、1 文字入力します。

[指定形式]

```
#include <stdio.h>
long getchar(void);
```

[戻り値]

正常：ファイルの終了の時：EOF
　　ファイルの終了でない時：入力した文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。
getchar 関数は標準入力ファイル (stdin) から 1 文字入力します。
getchar 関数は、通常入力した 1 文字を返しますが、ファイルの終了やエラー発生の時は EOF を返します。また、ファイルの終了の時には、そのファイルに対するファイル終了指示子が設定されます。

gets

標準入力ファイル (stdin) から文字列を入力します。

[指定形式]

```
#include <stdio.h>
char *gets(char *s);
```

[引数]

s　文字列を入力する記憶域へのポインタ

[戻り値]

正常：ファイルの終了の時：NULL
　　ファイルの終了でない時：s
異常：NULL

[備考]

gets 関数は、標準入力ファイル (stdin) から、s で始まる記憶域へ文字列を入力します。
gets 関数は、ファイルの終了か、改行文字を入力するまで文字を入力し、改行文字の代わりにヌル文字を付け加えます。
gets 関数は、通常文字列を入力する記憶域へのポインタ s を返しますが、標準入力ファイルの終了やエラー発生の時は、NULL を返します。
標準入力ファイルが終了した時は、s が指す記憶域の内容は変化しませんが、エラー発生の時は s が指す記憶域の内容は保証しません。

putc

ストリーム入出力用ファイルへ 1 文字出力します。

[指定形式]

```
#include <stdio.h>
long putc(long c, FILE *fp);
```

[引数]

c　出力する文字
fp　ファイルポインタ

[戻り値]

正常：出力した文字
異常：EOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
putc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力ファイルへ出力します。
putc 関数は、通常出力した文字 c を返しますが、エラー発生の時は EOF を返します。

putchar

標準出力ファイル (stdout) へ 1 文字出力します。

[指定形式]

```
#include <stdio.h>
long putchar(long c);
```

[引数]

c 出力する文字

[戻り値]

正常：出力した文字
異常：EOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
putchar 関数は、文字 c を標準出力ファイル (stdout) へ出力します。putchar マクロは、通常出力した文字 c を返しますが、エラー発生の時は EOF を返します。

puts

標準出力ファイル (stdout) へ文字列を出力します。

[指定形式]

```
#include <stdio.h>
long puts(const char *s);
```

[引数]

s　出力する文字列へのポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

puts 関数は、s の指す文字列を標準出力ファイル (stdout) へ出力します。この時、文字列の終了を示す文字は出力されず、代わりに改行文字を出力します。
puts 関数は、通常 0 を返しますが、エラー発生の時は 0 以外の値を返します。

ungetc

ストリーム入出力用ファイルへ 1 文字を戻します。

[指定形式]

```
#include <stdio.h>
long ungetc(long c, FILE *fp);
```

[引数]

c　戻す文字
fp　ファイルポインタ

[戻り値]

正常：戻した文字
異常：EOF

[備考]

ungetc 関数は、文字 c をファイルポインタ fp の示すストリーム入出力用ファイルへ戻します。
また、ここで戻された文字は、fflush、fseek、rewind 関数を呼び出さなければ次の入力データとなります。
ungetc 関数は、通常戻した文字 c を返しますが、エラー発生の時は EOF を返します。
ungetc 関数が fflush、fseek、rewind 関数を実行することなく 2 回以上呼び出された時の動作は保証しません。また、ungetc 関数が実行されるとファイルに対する現在の位置指示子が一つ戻されますが、この位置指示子がすでにファイルの先頭に位置している時は、位置指示子は保証しません。

fread

ストリーム入出力用ファイルから、指定した記憶域にデータを入力します。

[指定形式]

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n, FILE *fp);
```

[引数]

ptr　データを入力する記憶域へのポインタ
size　1 メンバのバイト数
n　入力するメンバの数
fp　ファイルポインタ

[戻り値]

size もしくは n が 0 の時：0
size,n がともに 0 でない時：入力に成功したメンバ数

[備考]

fread 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルから ptr が指す記憶域に size で指定したバイト数を 1 メンバとしたデータを n メンバ入力します。この時、ファイルに対する位置指示子は入力したバイト数分進められます。

fread 関数は、実際に入力に成功したメンバ数を返しますので、通常 n と同じ値になります。しかし、ファイルが終了した時やエラー発生の時は、それまで入力に成功したメンバ数を返しますので、n より小さな値となります。ファイルの終了かエラー発生かの区別は、ferror、feof 関数を用いて行ってください。

size もしくは n が 0 の時、リターン値として 0 を返し、ptr の指す記憶域の内容は変化しません。また、エラーが発生した時、または、メンバの途中までしか入力できなかった時は、そのファイルの位置指示子は保証しません。

fwrite

メモリ領域からストリーム入出力用ファイルにデータを出力します。

[指定形式]

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp);
```

[引数]

ptr　出力するデータを格納している記憶域へのポインタ
size　1 メンバのバイト数
n　出力するメンバの数
fp　ファイルポインタ

[戻り値]

出力に成功したメンバ数

[備考]

fwrite 関数は、ptr の指す記憶域から、ファイルポインタ fp の示すストリーム入出力用ファイルに、size で指定したバイト数を 1 メンバとしたデータを n メンバ出力します。
この時、ファイルに対する位置指示子は出力したバイト数進められます。
fwrite 関数は、実際に出力に成功したメンバ数を返しますので、通常 n と同じ値になります。しかし、エラー発生の時はそれまで出力に成功したメンバ数を返しますので、n より小さな値となります。
エラー発生の時、そのファイルの位置指示子は保証しません。

fseek

ストリーム入出力用ファイルの現在の読み書き位置を移動します。

[指定形式]

```
#include <stdio.h>
long fseek(FILE *fp, long offset, long type);
```

[引数]

fp　ファイルポインタ
offset　オフセットの種類で指定された位置からのオフセット
type　オフセットの種類

[戻り値]

正常：0
異常：0 以外

[備考]

fseek 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置をオフセットの種類 type で指定した場所から offset バイト先の位置に移動します。
オフセットの種類を表 7.14 に示します。
fseek 関数は、通常は 0 を返しますが、不適当な要求に対しては 0 以外の値を返します。

表 7.14　オフセットの種類

	オフセットの種類	意味
1	SEEK_SET	ファイルの先頭から offset バイト先の位置に移動します。この時、offset で指定する値は 0 か正でなければなりません。
2	SEEK_CUR	ファイルの現在位置から offset バイト先の位置に移動します。この時、offset で指定する値が正ならばファイルの後方に、負ならばファイルの先頭に向かって移動します。
3	SEEK_END	ファイルの終わりから offset バイト先の位置に移動します。この時 offset で指定する値は 0 か負でなければなりません。

テキストファイルの時は、オフセットの種類は SEEK_SET で、かつ offset は 0 かそのファイルに対する ftell 関数によって返された値でなければなりません。また、fseek 関数を呼び出すことによって ungetc 関数の効果はなくなりますので注意が必要です。

ftell

ストリーム入出力用ファイルの現在の読み書き位置を求めます。

[指定形式]

```
#include <stdio.h>
long ftell(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

現在の位置指示子の位置 (テキストファイル)
ファイルの先頭から現在位置までのバイト数 (バイナリファイル)

[備考]

ftell 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置を求めます。
ftell 関数は、バイナリファイルの時、ファイルの先頭から現在位置までのバイト数を返しますが、テキストファイルの時は、ここで返した値が fseek 関数で使用できるように処理系定義の値を位置指示子の位置として返します。
ftell 関数を 2 回テキストファイルに適用した時、そのリターン値の差が実際のファイル上の隔たりを表すことにはなりません。

rewind

ストリーム入出力用ファイルの現在の読み書き位置を、ファイルの先頭に移動します。

[指定形式]

```
#include <stdio.h>
void rewind(FILE *fp);
```

[引数]

fp　ファイルポインタ

[備考]

rewind 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルの現在の読み書き位置をファイルの先頭に移動します。
また、rewind 関数は、そのファイルに対する終了指示子とエラー指示子をクリアします。
rewind 関数を呼び出すことによって、ungetc 関数の効果はなくなりますので、注意が必要です。

clearerr

ストリーム入出力用ファイルのエラー状態をクリアします。

[指定形式]

#include <stdio.h>

[引数]

fp　ファイルポインタ

[備考]

clearerr 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルに対するエラー指示子と終了指示子をクリアします。

feof

ストリーム入出力用ファイルが終わりであるかどうかを判定します。

[指定形式]

```
#include <stdio.h>
long feof(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

ファイルが終わりの時：0 以外
ファイルが終わりでない時：0

[備考]

feof 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルが終了したかどうかを判定します。
feof 関数は、指定したストリーム入出力用ファイルに対するファイル終了指示子を調べ、設定されていればファイルが終わりであるとして、0 以外の値を返します。設定されていなければ、ファイルはまだ終わりではないとして 0 を返します。

ferror

ストリーム入出力用ファイルがエラー状態であるかどうかを判定します。

[指定形式]

```
#include <stdio.h>
long ferror(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

ファイルがエラー状態の時：0 以外
ファイルがエラー状態でない時：0

[備考]

ferror 関数は、ファイルポインタ fp の示すストリーム入出力用ファイルがエラー状態であるかどうかを判定します。
ferror 関数は、指定したストリーム入出力用ファイルに対するエラー指示子を調べ、設定されていれば、エラー状態にあるとして 0 以外の値を返します。設定されていなければ、エラー状態ではないとして 0 を返します。

perror

標準エラーファイル (stderr) に、エラー番号に対応したエラーメッセージを出力します。

[指定形式]

```
#include <stdio.h>
void perror(const char *s);
```

[引数]

s　エラーメッセージへのポインタ

[備考]

perror 関数は標準エラーファイル (stderr) へ s で示されるエラーメッセージと errno とを対応させ出力します。
出力するメッセージは、もし、s が NULL でなく、s の指す文字列がヌル文字でなければ、s の指す文字列にコロンと空白とその後に処理系定義のエラーメッセージを続け、最後に改行文字を付けた形式で出力されます。

7.4.12 <stdlib.h>

C プログラムでの標準的処理を行う関数を定義しています。
以下のマクロは、処理系定義です。

種別	定義名	説明
型 (マクロ)	onexit_t	onexit 関数で登録する関数の返す型および onexit 関数のリターン値の型です。
	div_t	div 関数のリターン値の構造体の型です。
	ldiv_t	ldiv 関数のリターン値の構造体の型です。
	lldiv_t	lldiv 関数のリターン値の構造体の型です。
定数 (マクロ)	RAND_MAX	rand 関数において生成する擬似乱数整数の最大値です。
	EXIT_SUCCESS	成功終了状態を表します。
関数	atof	数を表現する文字列を double 型の浮動小数点値に変換します。
	atoi	10 進数を表現する文字列を int 型の整数値に変換します。
	atol	10 進数を表現する文字列を long 型の整数値に変換します。
	atoll	10 進数を表現する文字列を long long 型の整数値に変換します。
	strtod	数を表現する文字列を double 型の浮動小数点値に変換します。
	strtof	数を表現する文字列を float 型の浮動小数点値に変換します。
	strtold	数を表現する文字列を long double 型の浮動小数点値に変換します。
	strtol	数を表現する文字列を long 型の整数値に変換します。
	strtoul	数を表現する文字列を unsigned long 型の整数値に変換します。
	strtoll	数を表現する文字列を long long 型の整数値に変換します。
	strtoull	数を表現する文字列を unsigned long long 型の整数値に変換します。
	rand	0 から RAND_MAX の間の擬似乱数整数を生成します。
	srand	rand 関数で生成する擬似乱数列の初期値を設定します。
	calloc	記憶域を割り当てて、すべての割り当てられた記憶域を 0 で初期化します。
	free	指定された記憶域を解放します。
	malloc	記憶域を割り当てます。
	realloc	記憶域の大きさを指定された大きさに変更します。
	bsearch	2 分割検索を行います。
	qsort	ソートを行います。
	abs	int 型整数の絶対値を計算します。
	div	int 型整数の除算の商と余りを計算します。
	labs	long 型整数の絶対値を計算します。
	ldiv	long 型整数の除算の商と余りを計算します。
	llabs	long long 型整数の絶対値を計算します。
	lldiv	long long 型整数の除算の商と余りを計算します。

処理系定義仕様

	項目	コンパイラの仕様
1	calloc,malloc,realloc 関数でサイズが 0 のときの動作	NULL を返します。

atof

数を表現する文字列を、double 型の浮動小数点値に変換します。

[指定形式]

```
#include <stdlib.h>
double atof(const char *nptr);
```

[引数]

nptr　変換する数を表現する文字列のポインタ

[戻り値]

変換された double 型の浮動小数点値

[備考]

変換後の値がオーバフロー / アンダフローをおこした時は errno を設定します。
変換は、浮動小数点型の形式に合わない最初の文字までに対して行います。
atof 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtod 関数を使用してください。

atoi

10 進数を表現する文字列を、int 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
long atoi(const char *nptr);
```

[引数]

nptr　変換する数を表現する文字列のポインタ

[戻り値]

変換された int 型の整数値

[備考]

変換後の値がオーバフローをおこした時は errno を設定します。
変換は、10 進数の形式に合わない最初の文字までに対して行います。
atoi 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtol 関数を使用してください。

atol

10 進数を表現する文字列を、long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
long atol(const char *nptr);
```

[引数]

nptr　変換する数を表現する文字列のポインタ

[戻り値]

変換された long 型の整数値

[備考]

変換後の値がオーバフローをおこした時は errno を設定します。
変換は、10 進数の形式に合わない最初の文字までに対して行います。
atol 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtol 関数を使用してください。

atoll

10 進数を表現する文字列を、long long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
long long atoll (const char *nptr);
```

[引数]

nptr　変換する数を表現する文字列のポインタ

[戻り値]

変換された long long 型の整数値

[備考]

変換後の値がオーバフローをおこした時は errno を設定します。
変換は、10 進数の形式に合わない最初の文字までに対して行います。
atoll 関数は、オーバフロー等のエラーが生じた場合、結果の値を保証しません。エラー時に保証された値を得たい場合は、strtoll 関数を使用してください。

strtod

数を表現する文字列を double 型の浮動小数点値に変換します。

[指定形式]

```
#include <stdlib.h>
double strtod(const char *nptr, char **endptr);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

[戻り値]

正常：nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
　　nptr が指している文字列が浮動小数点型を構成する文字で始まっている時：変換された double 型の浮動小数点値
異常：変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VAL
　　変換後の値がアンダフローの時：0

[備考]

変換後の値がオーバフロー / アンダフローをおこした時は errno を設定します。
strtod 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを double 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

strtof

数を表現する文字列を float 型の浮動小数点値に変換します。

[指定形式]

```
#include <stdlib.h>
float strtof(const char *nptr, char **endptr);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

[戻り値]

正常：nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
　　nptr が指している文字列が浮動小数点型を構成する文字で始まっている時：変換された float 型の浮動小数点値
異常：変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VALF
　　変換後の値がアンダフローの時：0

[備考]

変換後の値がオーバフロー / アンダフローをおこした時は errno を設定します。
strtof 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを float 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

strtold

数を表現する文字列を long double 型の浮動小数点値に変換します。

[指定形式]

```
#include <stdlib.h>
long double strtold(const char *nptr, char **endptr);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

[戻り値]

正常：nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
　　nptr が指している文字列が浮動小数点型を構成する文字で始まっている時：変換された long double 型の浮動小数点値
異常：変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VALL
　　変換後の値がアンダフローの時：0

[備考]

変換後の値がオーバフロー / アンダフローをおこした時は errno を設定します。
strtold 関数は、最初の数字もしくは小数点から浮動小数点値を構成しない文字の直前までを long double 型の浮動小数点値に変換します。ただし、指数部も小数点も現われなかった時は、小数点は文字列の最後の数字の後に続くと仮定されます。endptr の指す領域には、浮動小数点型を構成しない最初の文字へのポインタを設定します。数字を読み込む前に浮動小数点型を構成しない文字がある場合は nptr の値を設定します。endptr が NULL の場合、この設定は行われません。

strtol

数を表現する文字列を long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
long strtol(const char *nptr, char **endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された long 型の整数値
異常：変換後の値がオーバフローの時：変換する文字列の符号に従って LONG_MAX あるいは LONG_MIN

[備考]

変換後の値がオーバフローをおこした時は、errno を設定します。
strtol 関数は、最初の数字から整数を構成しない最初の文字の前までを long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「3.1.3(4) 整数」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A) から z(もしくは Z) までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X) も無視されます。

strtoul

数を表現する文字列を unsigned long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
unsigned long strtoul (const char *nptr, char **endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された unsigned long 型の整数値
異常：変換後の値がオーバフローの時：ULONG_MAX

[備考]

変換後の値がオーバフローをおこした時は、errno を設定します。
strtoul 関数は、最初の数字から整数を構成しない最初の文字の前までを unsigned long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「3.1.3(4) 整数」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A) から z(もしくは Z) までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X) も無視されます。

strtoll

数を表現する文字列を long long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
long long strtoll (const char *nptr, char **endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された long long 型の整数値
異常：変換後の値がオーバフローの時：変換する文字列の符号に従って LLONG_MAX あるいは LLONG_MIN

[備考]

変換後の値がオーバフローをおこした時は、errno を設定します。
strtoll 関数は、最初の数字から整数を構成しない最初の文字の前までを long long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「3.1.3(4) 整数」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A) から z(もしくは Z) までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X) も無視されます。

strtoull

数を表現する文字列を unsigned long long 型の整数値に変換します。

[指定形式]

```
#include <stdlib.h>
unsigned long long strtoull (const char *nptr, char **endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された unsigned long long 型の整数値
異常：変換後の値がオーバフローの時：ULLONG_MAX

[備考]

変換後の値がオーバフローをおこした時は、errno を設定します。
strtoull 関数は、最初の数字から整数を構成しない最初の文字の前までを unsigned long long 型の整数値に変換します。
endptr の指す記憶域に、整数を構成しない最初の文字へのポインタを設定します。最初の数字を読み込む前に整数を構成しない文字がある場合は nptr の値を設定します。endptr が NULL 場合、この設定は行われません。
base の値が 0 の時は、「3.1.3(4) 整数」の規則に従って変換されます。base の値が 2 から 36 の間の時は、変換する時の基数を示しています。ここで変換する文字列中の a(もしくは A) から z(もしくは Z) までの文字は、10 から 35 の値に対応付けられます。base の値より大きいか等しい文字が、変換する文字列の中にある時は、そこで変換処理を終了します。また、符号の後にある 0 は、変換の時は無視され、また、base が 16 の時の 0x(もしくは 0X) も無視されます。

rand

0 から RAND_MAX の間の擬似乱数整数を生成します。

[指定形式]

```
#include <stdlib.h>
long rand(void);
```

[戻り値]

擬似乱数整数値

srand

rand 関数で生成する擬似乱数列の初期値を設定します。

[指定形式]

```
#include <stdlib.h>
void srand(unsigned long seed);
```

[引数]

seed　擬似乱数列生成の初期値

[備考]

srand 関数は、rand 関数が擬似乱数列を生成するための初期値を設定します。したがって、rand 関数で擬似乱数値を生成している時に、再度 srand 関数で、同じ値の初期値を設定すると、擬似乱数列はくり返し生成されることになります。

rand 関数が srand 関数より先に呼ばれた時は、擬似乱数列の生成の初期値として 1 が設定されます。

calloc

記憶域を割り当てて、すべての割り当てられた記憶域を 0 で初期化します。

[指定形式]

```
#include <stdlib.h>
void *calloc(size_t nelem, size_t elsize);
```

[引数]

nelem　要素の数
elsize　一つの要素の占めるバイト数

[戻り値]

正常：割り当てられた記憶域の先頭のアドレス
異常：記憶域の割り当てができなかった時、または引数のいずれかが 0 の時：NULL

[備考]

elsize バイト単位の記憶域を nelem 個記憶域に割り当てます。また、その割り当てられた記憶域のすべてのビットは 0 で初期化されます。

CC-RX には、記憶域に対する不正な操作を検出するためのセキュリティ機能があります。詳細は、「2.5.4 ライブラリジェネレータ・オプション」の -secure_malloc オプションを参照してください。

free

指定された記憶域を解放します。

[指定形式]

```
#include <stdlib.h>
void free(void *ptr);
```

[引数]

ptr　解放する記憶域のアドレス

[備考]

ptr が指す記憶域を解放し、再度割り当てて使用することを可能とします。ptr が NULL であれば何もしません。

解放しようとした記憶域が、calloc、malloc、realloc 関数で割り当てられた記憶域でない時、または、すでに free、realloc 関数によって解放されていた時の動作は保証しません。また、解放された後の記憶域を参照した時の動作も保証しません。

CC-RX には、記憶域に対する不正な操作を検出するためのセキュリティ機能があります。詳細は、「2.5.4 ライブラリジェネレータ・オプション」の -secure_malloc オプションを参照してください。

malloc

記憶域を割り当てます。

[指定形式]

```
#include <stdlib.h>
void *malloc(size_t size);
```

[引数]

size　割り当てる記憶域のバイト数

[戻り値]

正常：割り当てられた記憶域の先頭アドレス
異常：記憶域の割り当てができなかった時、または size が 0 の時：NULL

[備考]

size で示されるバイトの分だけ記憶域を割り当てます。
CC-RX には、記憶域に対する不正な操作を検出するためのセキュリティ機能があります。詳細は、「2.5.4 ライブラリジェネレータ・オプション」の -secure_malloc オプションを参照してください。

realloc

記憶域の大きさを指定された大きさに変更します。

[指定形式]

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

[引数]

ptr　変更する記憶域の先頭アドレス
size　変更後の記憶域のバイト数

[戻り値]

正常：変更した記憶域の先頭アドレス
異常：記憶域の割り当てができなかった時、または size が 0 の時：NULL

[備考]

ptr の指す記憶域の大きさを size で示されるバイト分の大きさの記憶域に変更します。もし、新しく割り当てられた記憶域の大きさが、変更前の記憶域の大きさより小さい時は、新しく割り当てられた記憶域の大きさまでの内容は変化しません。

ptr が calloc、malloc、realloc 関数で割り当てられた記憶域へのポインタでない時、またはすでに free、realloc 関数によって解放されている記憶域へのポインタの時、動作はされません。

CC-RX には、記憶域に対する不正な操作を検出するためのセキュリティ機能があります。詳細は、「2.5.4 ライブラリジェネレータ・オプション」の -secure_malloc オプションを参照してください。

bsearch

二分割検索を行います。

[指定形式]

```
// C 言語の場合
#include <stdlib.h>
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, long (*compar)(const void *, const void *));

// C++ 言語 /EC++ 言語の場合
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
```

[引数]

key　検索するデータへのポインタ
base　検索対象となるテーブルへのポインタ
nmemb　検索対象のメンバの数
size　検索対象のメンバのバイト数
compar　比較を行う関数へのポインタ

[戻り値]

一致するメンバが検索できた時：一致したメンバへのポインタ
一致するメンバが検索できなかった時：NULL

[備考]

key の指すデータと一致するメンバを、base の指すテーブルの中で二分割検索法によって検索します。比較を行う関数は、比較する 2 つのデータへのポインタ p1(第 1 引数)、p2(第 2 引数) を受け取り、以下の仕様に従って結果を返してください。

*p1<*p2 の時、負の値を返します。
*p1==*p2 の時、0 を返します。
*p1>*p2 の時、正の値を返します。

検索対象となる各メンバは、昇順に並んでいる必要があります。

qsort

ソートを行います。

[指定形式]

```
// C 言語の場合
#include <stdlib.h>
void qsort(const void *base, size_t nmemb, size_t size, long (*compar)(const void *, const void *));

// C++ 言語 /EC++ 言語の場合
void *bsearch(const void *key, const void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
```

[引数]

base　ソート対象となるテーブルへのポインタ
nmemb　ソート対象のメンバの数
size　ソート対象のメンバのバイト数
compar　比較を行う関数へのポインタ

[備考]

base の指すテーブルのデータをソートします。データの並べる順序は、比較を行う関数へのポインタによって指定します。この関数は、比較する 2 つのデータへのポインタ p1(第 1 引数)、p2(第 2 引数) を受け取り、以下の仕様に従って結果を返してください。

*p1<*p2 の時、負の値を返します。
*p1==*p2 の時、0 を返します。
*p1>*p2 の時、正の値を返します。

abs

int 型整数の絶対値を求めます。

[指定形式]

```
#include <stdlib.h>
long abs(long i);
```

[引数]

i　絶対値を求める整数

[戻り値]

i の絶対値

[備考]

i の絶対値を求めた結果、int 型整数値として表現できない時の動作は保証しません。

div

int 型整数の除算の商と余りを計算します。

[指定形式]

```
#include <stdlib.h>
div_t div(long numer, long denom);
```

[引数]

numer　被除数
denom　除数

[戻り値]

numer を denom で除算した結果の商と余り

[備考]

型 div_t は次のように定義されています。

```
typedef struct {
    long quot;
    long rem;
} div_t;
```

labs

long 型整数の絶対値を計算します。

[指定形式]

```
#include <stdlib.h>
long labs(long j);
```

[引数]

j　絶対値を求める整数

[戻り値]

j の絶対値

[備考]

j の絶対値を求めた結果、long 型の整数値として表現できない時の動作は保証しません。

ldiv

long 型整数の除算の商と余りを計算します。

[指定形式]

```
#include <stdlib.h>
ldiv_t ldiv(long numer, long denom);
```

[引数]

numer　被除数
denom　除数

[戻り値]

numer を denom で除算した結果の商と余り

[備考]

型 ldiv_t は次のように定義されています。

```
typedef struct {
    long quot;
    long rem;
} ldiv_t;
```

llabs

long long 型整数の絶対値を計算します。

[指定形式]

```
#include <stdlib.h>
long long llabs(long long j);
```

[引数]

j　絶対値を求める整数

[戻り値]

j の絶対値

[備考]

j の絶対値を求めた結果、long long 型の整数値として表現できない時の動作は保証しません。

lldiv

long long 型整数の除算の商と余りを計算します。

[指定形式]

```
#include <stdlib.h>
lldiv_t lldiv(long long numer, long long denom);
```

[引数]

numer　被除数
denom　除数

[戻り値]

numer を denom で除算した結果の商と余り

[備考]

型 lldiv_t は次のように定義されています。

```
typedef struct {
    long long quot;
    long long rem;
} lldiv_t;
```

7.4.13 <string.h>

文字配列の操作に必要な種々の関数を定義します。

種別	定義名	説明
関数	memcpy	複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。
	strcpy	複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。
	strncpy	複写元の文字列を指定された文字数分、複写先の記憶域に複写します。
	strcat	文字列の後に、文字列を連結します。
	strncat	文字列に文字列を指定した文字数分、連結します。
	memcmp	指定された 2 つの記憶域の比較を行います。
	strcmp	指定された 2 つの文字列を比較します。
	strncmp	指定された 2 つの文字列を指定された文字数分まで比較します。
	memchr	指定された記憶域において、指定された文字が最初に現われる位置を検索します。
	strchr	指定された文字列において、指定された文字が最初に現われる位置を検索します。
	strcspn	指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。
	strpbrk	指定された文字列において、別に指定された文字列中の文字が最初に現われる位置を検索します。
	strrchr	指定された文字列において指定された文字が最後に現われる位置を検索します。
	strspn	指定された文字列を先頭から調べ別に指定した文字列中の文字が先頭から何文字続くかを求めます。
	strstr	指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。
	strtok	指定した文字列をいくつかの字句に切り分けます。
	memset	指定された記憶域の先頭から指定された文字を指定された文字数分設定します。
	strerror	エラーメッセージを設定します。
	strlen	文字列の文字数を計算します。
	memmove	複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。複写元と複写先の記憶域が重なっていても、正しく複写されます。

処理系定義仕様

	項目	コンパイラの仕様
1	strerror 関数が返すエラーメッセージの内容	「10.5.6 C 標準ライブラリ関数のエラーメッセージ」を参照してください。

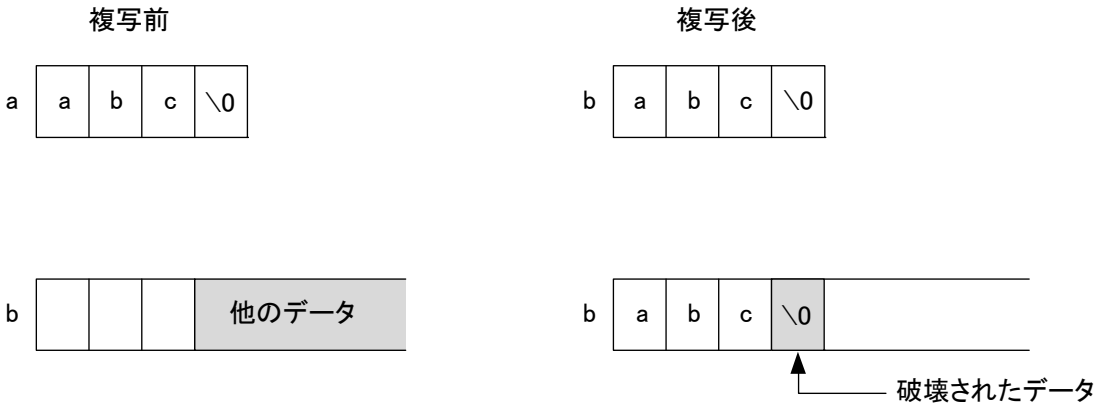
本標準インクルードファイル内で定義されている関数を使用する時は、以下の 2 つの事項に注意する必要があります。

(1) 文字列の複写を行う時、複写先の領域が複写元の領域よりも小さい場合、動作は保証しませんので注意が必要です。

例

```
char a[]="abc";
char b[3];
     :
     :
strcpy(b,a);
```

この場合、配列 a のサイズは (ヌル文字を含めて)4 バイトです。したがって、strcpy 関数によって複写を行うと、配列 b の領域以外のデータを書き換えることになります。

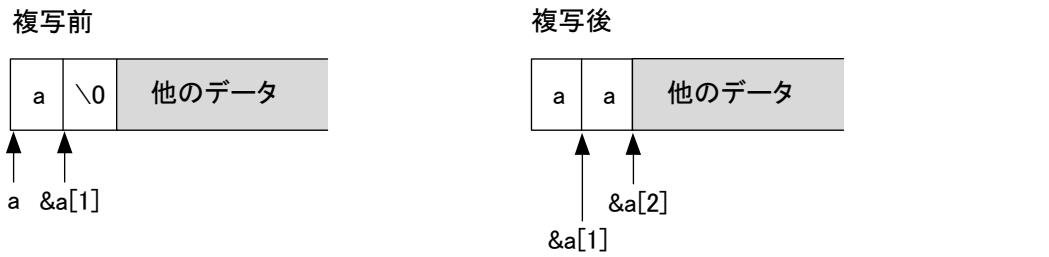


(2) 文字列の複写を行う時、複写元の領域と複写先の領域が重なっていると正しい動作は保証しませんので注意が必要です。

例

```
int a[]="a";
     :
     :
strcpy(&a[1], a);
     :
     :
```

この場合、複写元の文字列がヌル文字に達する以前に、ヌル文字の上に文字 'a' を書き込むことになります。したがって、複写元の文字列のデータに続くデータを書き換えることになります。



以下次々に文字を複写し続けます。

memcpy

複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。

[指定形式]

```
#include <string.h>
void *memcpy(void *s1, const void *s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の記憶域へのポインタ
n　複写する文字数

[戻り値]

s1 の値

strcpy

複写元の文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。

[指定形式]

```
#include <string.h>
char *strcpy(char *s1, const char *s2);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の文字列へのポインタ

[戻り値]

s1 の値

strncpy

複写元の文字列を指定された文字数分、複写先の記憶域に複写します。

[指定形式]

```
#include <string.h>
char *strncpy(char *s1, const char *s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の文字列へのポインタ
n　複写する文字数

[戻り値]

s1 の値

[備考]

s2 で指された文字列の先頭から最高 n 文字を s1 で指される記憶域に複写します。s2 で指定された文字列の文字数が n 文字より短い時は、n 文字になるまでヌル文字が付加されます。
s2 で指された文字列の文字数が n 文字より長い時は、s1 に複写された文字列はヌル文字で終了しないことになります。

strcat

文字列の後に、文字列を連結します。

[指定形式]

```
#include <string.h>
char *strcat(char *s1, const char *s2);
```

[引数]

s1　連結される文字列へのポインタ
s2　連結する文字列へのポインタ

[戻り値]

s1 の値

[備考]

s1 で指された文字列の最後に、s2 で指された文字列を連結します。この時、s2 の指す文字列の最後を示すヌル文字も複写します。また、s1 で指された文字列の最後のヌル文字は削除されます。

strncat

文字列に文字列を指定した文字数分連結します。

[指定形式]

```
#include <string.h>
char *strncat(char *s1, const char *s2, size_t n);
```

[引数]

s1　連結される文字列へのポインタ
s2　連結する文字列へのポインタ
n　連結する文字数

[戻り値]

s1 の値

[備考]

s2 で指された文字列の先頭から最高 n 文字を s1 で指された文字列の最後に付加します。s1 で指された文字列の最後のヌル文字は s2 の先頭文字で置き換えられます。
また、連結された後の文字列の最後には、必ずヌル文字が付加されます。

memcmp

指定された 2 つの記憶域の内容を比較します。

[指定形式]

```
#include <string.h>
long memcmp(const void *s1, const void *s2, size_t n);
```

[引数]

s1　比較される記憶域へのポインタ
s2　比較する記憶域へのポインタ
n　比較する記憶域の文字数

[戻り値]

s1 で指された記憶域 >s2 で指された記憶域の時：正の値
s1 で指された記憶域 ==s2 で指された記憶域の時：0
s1 で指された記憶域 <s2 で指された記憶域の時：負の値

[備考]

s1 で指された記憶域と s2 で指された記憶域の、最初の n 文字分の内容を比較します。
この比較は処理系定義です。

strcmp

指定された 2 つの文字列の内容を比較します。

[指定形式]

```
#include <string.h>
long strcmp(const char *s1, const char *s2);
```

[引数]

s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ

[戻り値]

s1 で指された文字列 >s2 で指された文字列の時：正の値
s1 で指された文字列 ==s2 で指された文字列の時：0
s1 で指された文字列 <s2 で指された文字列の時：負の値

[備考]

s1 で指された文字列と、s2 で指された文字列の内容を比較し、その結果をリターン値として設定します。
この比較は処理系定義です。

strncmp

指定された 2 つの文字列を指定された文字分まで比較します。

[指定形式]

```
#include <string.h>
long strncmp(const char *s1, const char *s2, size_t n);
```

[引数]

s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ
n　比較する文字数の最大値

[戻り値]

s1 で指された文字列 >s2 で指された文字列の時：正の値
s1 で指された文字列 ==s2 で指された文字列の時：0
s1 で指された文字列 <s2 で指された文字列の時：負の値

[備考]

s1 で指された文字列と、s2 で指された文字列を最初の n 文字以内の範囲で、その内容を比較します。
この比較は処理系定義です。

memchr

指定された記憶域において、指定された文字が最初に現われる位置を検索します。

[指定形式]

```
#include <string.h>
void *memchr(const void *s, long c, size_t n);
```

[引数]

s 検索を行う記憶域へのポインタ
c 検索する文字
n 検索を行う文字数

[戻り値]

検索の結果見つかった時：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

s で指定された記憶域の先頭から n 文字の中で最初に現われた c の文字と同一文字の位置へのポインタをリターン値として返します。

strchr

指定された文字列において、指定された文字が最初に現われる位置を検索します。

[指定形式]

```
#include <string.h>
char *strchr(const char *s, long c);
```

[引数]

s 検索を行う文字列へのポインタ
c 検索する文字

[戻り値]

検索の結果見つかった時：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

s で指定された文字列中で最初に現われた c の文字と同一文字へのポインタをリターン値として返します。
s によって指される文字列の終了を現わすヌル文字も検索の対象として含まれます。

strcspn

指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

[指定形式]

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
```

[引数]

s1　調べられる文字列へのポインタ
s2　s1 を調べるための文字列へのポインタ

[戻り値]

s2 が指す文字列を構成する文字以外の文字が構成される文字列 s1 の先頭からの長さ

[備考]

s2 が指す文字列を構成する文字以外の文字が、文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の文字数をリターン値として返します。
s2 によって指される文字列の終了を表すヌル文字は、s2 で指された文字列の一部とはみなされません。

strpbrk

指定された文字列内において、別に指定された文字列中の文字が最初に現われる位置を検索します。

[指定形式]

```
#include <string.h>
char *strpbrk(const char *s1, const char *s2);
```

[引数]

s1　検索を行う文字列へのポインタ
s2　s1 内で検索する文字を示す文字列へのポインタ

[戻り値]

検索の結果見つかった時：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

s1 で指された文字列において、s2 で指された文字列中の文字の一つが最初に現われる所を検索し、そのポインタをリターン値として返します。

strrchr

指定された文字列において、指定された文字が最後に現われる位置を検索します。

[指定形式]

```
#include <string.h>
char *strrchr(const char *s, long c);
```

[引数]

s 検索を行う文字列へのポインタ
c 検索する文字

[戻り値]

検索の結果見つかった時：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

s で指された文字列の中で c で指定する文字と同一の文字が最後に現われた位置へのポインタをリターン値として返します。
s によって指される文字列の終了を表すヌル文字も検索の対象として含まれます。

strspn

指定された文字列を先頭から調べ、別に指定した文字列中の文字が先頭から何文字続くかを求めます。

[指定形式]

```
#include <string.h>
size_t strspn(const char *s1, const char *s2);
```

[引数]

s1　調べられる文字列へのポインタ
s2　s1 を調べるための文字列へのポインタ

[戻り値]

s1 の先頭から、s2 で指定した文字が続いている文字数

[備考]

s2 が指す文字列を構成する文字が文字列として何文字続くかを s1 で指された文字列の先頭から調べ、その文字列の文字数をリターン値として返します。

strstr

指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。

[指定形式]

```
#include <string.h>
char *strstr(const char *s1, const char *s2);
```

[引数]

s1　検索を行う文字列へのポインタ
s2　検索する文字列へのポインタ

[戻り値]

検索の結果見つかったとき：見つけられた文字へのポインタ
検索の結果見つからなかったとき：NULL

[備考]

s1 で指された文字列において、s2 で指された文字列が最初に現われる所を検索し、そのポインタをリターン値として返します。

strtok

指定した文字列をいくつかの字句に切り分けます。

[指定形式]

```
#include <string.h>
char *strtok(char *s1, const char *s2);
```

[引数]

s1　いくつかの字句に切り分ける文字列へのポインタ
s2　文字列を切り分けるための文字からなる文字列へのポインタ

[戻り値]

字句に切り分けられた時：切り分けた字句の先頭へのポインタ
字句に切り分けられなかった時：NULL

[備考]

strtok 関数は文字列を切り分けるために連続的に呼び出されます。

(a)　最初の呼び出し時
s1 で指された文字列を先頭から s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。

(b)　2 回目以降の呼び出し時
以前に切り分けられた字句の次の文字から、s2 で指された文字列中の文字によって字句に切り分けます。その結果字句に切り分けられれば、その字句の先頭へのポインタを、分けられなければ NULL をリターン値として返します。
2 回目以降の呼び出しの時は、第 1 引数には NULL を指定します。また、s2 で指された文字列は呼び出しのたびに異なっていてもかまいません。切り出された字句の最後にはヌル文字が付きます。
strtok 関数の使用例を以下に示します。

例

```
1  #include <string.h>
2  static char s1[]="a@b,@c/@d";
3  char *ret;
4
5  ret=strtok(s1,"@");
6  ret=strtok(NULL,",@");
7  ret=strtok(NULL,"/@");
8  ret=strtok(NULL,"@");
```

【説明】
この例は、文字列「"a@b,@c/@d"」を strtok 関数を用いて a,b,c,d という字句に切り分けるプログラムを示しています。
2 行目で文字列 s1 に初期値として、文字列 "a@b,@c/@d" を設定しています。
5 行目では、「@」を区切り文字として字句を切り分けるため、strtok 関数を呼び出します。この結果、文字 'a' へのポインタがリターン値として得られ、文字 'a' の次の最初の区切り文字である「@」にヌル文字を埋め込みます。この結果、文字列 "a" が切り出されます。
以下、同一の文字列から次々に字句を切り出すために第 1 引数に NULL を指定して strtok 関数を呼び出します。
この結果、文字列 "b"、"c"、"d" が次々に切り出されます。

memset

指定された記憶域の先頭から、指定された文字を指定された文字数分設定します。

[指定形式]

```
#include <string.h>
void *memset(void *s, long c, size_t n);
```

[引数]

s 文字が設定される記憶域へのポインタ
c 設定する文字
n 設定する文字数

[戻り値]

s の値

[備考]

s で指された記憶域に n 文字分、文字 c を設定します。

strerror

エラー番号を指定して、それに対するエラーメッセージを返します。

[指定形式]

```
#include <string.h>
char *strerror(long s);
```

[引数]

s エラー番号

[戻り値]

エラー番号に対応するエラーメッセージ (文字列) へのポインタ

[備考]

エラー番号 s に対応するエラーメッセージへのポインタをリターン値として返します。
エラーメッセージの内容に関しては処理系定義です。
リターン値として返されたエラーメッセージを修正した時、動作は保証しません。

strlen

文字列の文字数を計算します。

[指定形式]

```
#include <string.h>
size_t strlen(const char *s);
```

[引数]

s 長さを求める文字列へのポインタ

[戻り値]

文字列の文字数

[備考]

s が指す文字列の終了を表すヌル文字は、文字列の文字数としては計算に入れません。

memmove

複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。また、複写元と複写先の記憶域が、重なっている部分があっても、複写元の重なっている部分を上書きする前に複写するので正しく複写されます。

[指定形式]

```
#include <string.h>
void *memmove(void *s1, const void *s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の記憶域へのポインタ
n　複写する文字数

[戻り値]

s1 の値

7.4.14 <complex.h>

各種の複素数計算を行います。float 型の複素数の場合は、定義名の最後に 'f'、long double 型の複素数の場合は、定義名の最後に 'l'、double 型の複素数の場合は、定義名が関数名になります。

種別	定義名	説明
関数	cacosf / cacos / cacosl <*-lang=c99*>	複素数逆余弦を計算します。
	casinf / casin / casinl <*-lang=c99*>	複素数逆正弦を計算します。
	catanf / catan / catanl <*-lang=c99*>	複素数逆正接を計算します。
	ccosf / ccos / ccosl <*-lang=c99*>	複素数余弦を計算します。
	csinf / csin / csinl <*-lang=c99*>	複素数正弦を計算します。
	ctanf / ctan / ctanl <*-lang=c99*>	複素数正接を計算します。
	cacoshf / cacosh / cacoshl <*-lang=c99*>	複素数逆双曲線余弦を計算します。
	casinhf / casinh / casinhl <*-lang=c99*>	複素数逆双曲線正弦を計算します。
	catanhf / catanh / catanhl <*-lang=c99*>	複素数逆双曲線正接を計算します。
	ccoshf / ccosh / ccoshl <*-lang=c99*>	複素数双曲線余弦を計算します。
	csinhf / csinh / csinhl <*-lang=c99*>	複素数双曲線正弦を計算します。
	ctanhf / ctanh / ctanhl <*-lang=c99*>	複素数双曲線正接を計算します。
	cexpf / cexp / cexpl <*-lang=c99*>	複素数自然対数の底 e の z 乗を計算します。
	clogf / clog / clogl <*-lang=c99*>	複素数自然対数を計算します。
	cabsf / cabs / cabsl <*-lang=c99*>	複素数絶対値を計算します。
	cpowf / cpow / cpowl <*-lang=c99*>	複素数べき乗を計算します。
	csqrtf / csqrt / csqrtl <*-lang=c99*>	複素数平方根を計算します。
	cargf / carg / cargl <*-lang=c99*>	偏角を計算します。
	cimagf / cimag / cimagl <*-lang=c99*>	虚部を計算します。
	conjf / conj / conjl <*-lang=c99*>	虚部の符号を反転させて複素共役を計算します。
	cprojf / cproj / cprojl <*-lang=c99*>	リーマン球面上への射影を計算します。
	crealf / creal / creall <*-lang=c99*>	実部を計算します。

cacosf / cacos / cacosl

複素数逆余弦を計算します。

[指定形式]

```
#include <complex.h>
float complex cacosf(float complex z);
double complex cacos(double complex z);
long double complex cacosl(long double complex z);
```

[引数]

z　複素数逆余弦を求める複素数

[戻り値]

z の複素数逆余弦値

[備考]

cacos 関数のリターン値の実軸方向の範囲は [0,π]、虚軸方向の範囲は無限の区間です。

casinf / casin / casinl

複素数逆正弦を計算します。

[指定形式]

```
#include <complex.h>
float complex casinf(float complex z);
double complex casin(double complex z);
long double complex casinl(long double complex z);
```

[引数]

z　複素数逆正弦を求める複素数

[戻り値]

z の複素数逆正弦値

[備考]

casin 関数のリターン値の実軸方向の範囲は [-π/2,π/2]、虚軸方向の範囲は無限の空間です。

catanf / catan / catanl

複素数逆正接を計算します。

[指定形式]

```
#include <complex.h>
float complex catanf(float complex z);
double complex catan(double complex z);
long double complex catanl(long double complex z);
```

[引数]

z　複素数逆正接を求める複素数

[戻り値]

z の複素数逆正接値

[備考]

catan 関数のリターン値の実軸方向の範囲は [-π/2,π/2]、虚軸方向の範囲は無限の空間です。

ccosf / ccos / ccosl

複素数余弦を計算します。

[指定形式]

```
#include <complex.h>
float complex ccosf(float complex z);
double complex ccos(double complex z);
long double complex ccosl(long double complex z);
```

[引数]

z　複素数余弦を求める複素数

[戻り値]

z の複素数余弦値

csinf / csin / csinl

複素数正弦を計算します。

[指定形式]

```
#include <complex.h>
float complex csinf(float complex z);
double complex csin(double complex z);
long double complex csinl(long double complex z);
```

[引数]

z　複素数正弦を求める複素数

[戻り値]

z の複素数正弦値

ctanf / ctan / ctanl

複素数正接を計算します。

[指定形式]

```
#include <complex.h>
float complex ctanf(float complex z);
double complex ctan(double complex z);
long double complex ctanl(long double complex z);
```

[引数]

z　複素数正接を求める複素数

[戻り値]

z の複素数正接値

cacoshf / cacosh / cacoshl

複素数逆双曲線余弦を計算します。

[指定形式]

```
#include <complex.h>
float complex cacoshf(float complex z);
double complex cacosh(double complex z);
long double complex cacoshl(long double complex z);
```

[引数]

z　複素数逆双曲線余弦を求める複素数

[戻り値]

z の複素数逆双曲線余弦値

[備考]

cacoshf 関数群のリターン値の範囲は [0, π] です。

casinhf / casinh / casinhl

複素数逆双曲線正弦を計算します。

[指定形式]

```
#include <complex.h>
float complex casinhf(float complex z);
double complex casinh(double complex z);
long double complex casinhl(long double complex z);
```

[引数]

z　複素数逆双曲線正弦を求める複素数

[戻り値]

z の複素数逆双曲線正弦値

catanhf / catanh / catanhl

複素数逆双曲線正接を計算します。

[指定形式]

```
#include <complex.h>
float complex catanhf(float complex z);
double complex catanh(double complex z);
long double complex catanhl(long double complex z);
```

[引数]

z　複素数逆双曲線正接を求める複素数

[戻り値]

z の複素数逆双曲線正接値

ccoshf / ccosh / ccoshl

複素数双曲線余弦を計算します。

[指定形式]

```
#include <complex.h>
float complex ccoshf(float complex z);
double complex ccosh(double complex z);
long double complex ccoshl(long double complex z);
```

[引数]

z　双曲線余弦を求める複素数

[戻り値]

z の複素数双曲線余弦値

csinhf / csinh / csinhl

複素数双曲線正弦を計算します。

[指定形式]

```
#include <complex.h>
float complex csinhf(float complex z);
double complex csinh(double complex z);
long double complex csinhl(long double complex z);
```

[引数]

z　双曲線正弦を求める複素数

[戻り値]

z の複素数双曲線正弦値

ctanhf / ctanh / ctanhl

複素数双曲線正接を計算します。

[指定形式]

```
#include <complex.h>
float complex ctanhf(float complex z);
double complex ctanh(double complex z);
long double complex ctanhl(long double complex z);
```

[引数]

z　双曲線正接を求める複素数

[戻り値]

z の複素数双曲線正接値

cexpf / cexp / cexpl

複素数の指数関数を計算します。

[指定形式]

```
#include <complex.h>
float complex cexpf(float complex z);
double complex cexp(double complex z);
long double complex cexpl(long double complex z);
```

[引数]

z　指数関数を求める複素数

[戻り値]

z の指数関数値

clogf / clog / clogl

複素数の自然対数を計算します。

[指定形式]

```
#include <complex.h>
float complex clogf(float complex z);
double complex clog(double complex z);
long double complex clogl(long double complex z);
```

[引数]

z　複素数自然対数を求める複素数

[戻り値]

正常：z の複素数自然対数値
異常：定義域エラーの時は、非数を返します。

[備考]

z の値が負の時、定義域エラーになります。
z の値が 0.0 の時、範囲エラーになります。
clog 関数群のリターン値の実軸方向の範囲は無限の区間、虚軸方向の範囲は [-iπ,+iπ] です。

cabsf / cabs / cabsl

複素数絶対値を計算します。

[指定形式]

```
#include <complex.h>
float cabsf(float complex z);
double cabs(double complex z);
long double cabsl(long double complex z);
```

[引数]

z　複素数絶対値を求める複素数

[戻り値]

z の複素数絶対値

cpowf / cpow / cpowl

複素数べき乗を計算します。

[指定形式]

```
#include <complex.h>
float complex cpowf(float complex x, float complex y);
double complex cpow(double complex x, double complex y);
long double complex cpowl(long double complex x, long double complex y);
```

[引数]

x　べき乗される値
y　べき乗する値

[戻り値]

正常：x の y 乗の値
異常：定義域エラーの時は、非数を返します。

[備考]

x の値が 0.0 で、かつ y の値が 0.0 以下の時、あるいは x の値が負で y の値が整数値でない時、定義域エラーになります。
cpow 関数群の第 1 仮引数に対する分岐切断線は負の実軸に沿っています。

csqrtf / csqrt / csqrtl

複素数平方根を計算します。

[指定形式]

```
#include <complex.h>
float complex csqrtf(float complex z);
double complex csqrt(double complex z);
long double complex csqrtl(long double complex z);
```

[引数]

z　平方根関数値を求める複素数

[戻り値]

正常：z の複素数平方根値
異常：定義域エラーの時は、非数を返します。

[備考]

z の値が負の値の時、定義域エラーになります。
csqrt 関数群の分岐分断線は負の実軸に沿っています。
csqrt 関数群のリターン値の領域は虚軸を含む右半平面です。

cargf / carg / cargl

偏角を計算します。

[指定形式]

```
#include <complex.h>
float cargf(float complex z);
double carg(double complex z);
long double cargl(long double complex z);
```

[引数]

z　偏角値を求める複素数

[戻り値]

z の偏角値

[備考]

carg 関数群の分岐切断線はは負の実軸に沿っています。
carg 関数群のリターン値の範囲は区間 [-π,+π] です。

cimagf / cimag / cimagl

虚部を計算します。

[指定形式]

```
#include <complex.h>
float cimagf(float complex z);
double cimag(double complex z);
long double cimagl(long double complex z);
```

[引数]

z　虚部を求める複素数

[戻り値]

実数としての z の虚部値

conjf / conj / conjl

虚部の符号を反転させて複素共役を計算します。

[指定形式]

```
#include <complex.h>
float complex conjf(float complex z);
double complex conj(double complex z);
long double complex conjl(long double complex z);
```

[引数]

z 複素共役値を求める複素数

[戻り値]

z の複素共役値

cprojf / cproj / cprojl

リーマン球面上への射影を計算します。

[指定形式]

```
#include <complex.h>
float complex cprojf(float complex z);
double complex cproj(double complex z);
long double complex cprojl(long double complex z);
```

[引数]

z　リーマン球面上への射影値を求める複素数

[戻り値]

リーマン球面上への z の射影値

crealf / creal / creall

実部を計算します。

[指定形式]

```
#include <complex.h>
float crealf(float complex z);
double creal(double complex z);
long double creall(long double complex z);
```

[引数]

z 実部値を求める複素数

[戻り値]

z の実部値

7.4.15 <fenv.h>

浮動小数点環境へアクセスします。
以下は、すべて処理系定義です。

種別	定義名	説明
型 (マクロ)	fenv_t	浮動小数点環境全体の型です。
	fexcept_t	浮動小数点状態フラグの型です。
定数 (マクロ)	FE_DIVBYZERO FE_INEXACT FE_INVALID FE_OVERFLOW FE_UNDERFLOW FE_ALL_EXCEPT	浮動小数点例外をサポートするときに定義されるマクロです。
	FE_DOWNWARD FE_TONEAREST FE_TOWARDZERO FE_UPWARD	浮動小数点数の丸め方向のマクロです。
	FE_DFL_ENV	プログラム既定の浮動小数点環境です。
関数	feclearexcept	浮動小数点例外のクリアを試みます。
	fegetexceptflag	浮動小数点フラグの状態のオブジェクトへの格納を試みます。
	feraiseexcept	浮動小数点例外の生成を試みます。
	fesetexceptflag	浮動小数点フラグのセットを試みます。
	fetestexcept	浮動小数点フラグがセットされているか確認します。
	fegetround	丸め方向を取得します。
	fesetround	丸め方向を設定します。
	fegetenv	浮動小数点環境の取得を試みます。
	feholdexcept	浮動小数点環境を保存し、浮動小数点状態フラグをクリアし、浮動小数点例外について無停止モードに設定します。
	fesetenv	浮動小数点環境の設定を試みます。
	feupdateenv	浮動小数点例外の自動記憶域への保存、浮動小数点環境の設定、保存していた浮動小数点例外の生成を試みます。

feclearexcept

浮動小数点例外のクリアを試みます。

[指定形式]

```
#include <fenv.h>
long feclearexcept(long e);
```

[引数]

e　浮動小数点例外

[戻り値]

正常：0
異常：0 以外

[備考]

本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fegetexceptflag

例外フラグの状態を取得します。

[指定形式]

```
#include <fenv.h>
long fegetexceptflag(fexcept_t *f, long e);
```

[引数]

f 例外フラグ状態の格納先を指すポインタ
e 状態を取得する例外フラグを表す値

[戻り値]

正常：0
異常：0 以外

[備考]

本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

feraiseexcept

浮動小数点例外の生成を試みます。

[指定形式]

```
#include <fenv.h>
long feraiseexcept(long e);
```

[引数]

e　生成を試みる例外を指す値

[戻り値]

正常：0
異常：0 以外

[備考]

feraiseexcept 関数が、“ オーバフロー ” 浮動小数点例外又は “ アンダフロー ” 浮動小数点例外を生成する際に “ 不正確結果 ” 浮動小数点例外を生成するかどうかは、処理系定義とします。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fesetexceptflag

例外フラグの状態を設定します。

[指定形式]

```
#include <fenv.h>
long fesetexceptflag(const fexcept_t *f, long e);
```

[引数]

f 例外フラグ状態の取得元を指すポインタ
e 状態を設定する例外フラグを表す値

[戻り値]

正常：0
異常：0 以外

[備考]

fesetexceptflag 関数を呼ぶ前にフラグ状態の取得元を fegetexceptflag 関数にて設定してください。
fesetexceptflag 関数は浮動小数点例外を生成せず、フラグの状態だけを設定します。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fetestexcept

例外フラグの状態を判定します。

[指定形式]

```
#include <fenv.h>
long fetestexcept(long e);
```

[引数]

e　状態を判定するフラグ (複数可) を表す値

[戻り値]

e と浮動小数点例外マクロのビット単位の論理和

[備考]

fetestexcept 関数は 1 回の呼び出しで複数個の浮動小数点例外を判定することができます。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fegetround

その時点の丸め方向を取得します。

[指定形式]

```
#include <fenv.h>
long fegetround(void);
```

[戻り値]

正常：0
異常：丸め方向マクロ値が存在しない又は丸め方向を決めることができない場合は負の値

[備考]

本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fesetround

その時点の丸め方向を設定します。

[指定形式]

```
#include <fenv.h>
#include <assert.h>
long fesetround(long rnd);
```

[戻り値]

成功した場合のみに 0

[備考]

fesetround 関数に丸め方向マクロの値と等しくない変更を要求した場合
丸め方向を変更しません。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fegetenv

浮動小数点環境を取得します。

[指定形式]

```
#include <fenv.h>
long fegetenv(fenv_t *f);
```

[引数]

f　浮動小数点環境格納先を指すポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

feholdexcept

浮動小数点環境を保存します。

[指定形式]

```
#include <fenv.h>
long feholdexcept(fenv_t *f);
```

[引数]

f 浮動小数点環境を指すポインタ

[戻り値]

成功した場合のみに 0

[備考]

feholdexcept 関数は浮動小数点関数環境保存時に浮動小数点状態フラグをクリアし、すべての浮動小数点例外について、無停止 (non-stop) モードを設定します。無停止モード設定時は浮動小数点例外発生時も実行を継続します。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

fesetenv

浮動小数点環境を設定します。

[指定形式]

```
#include <fenv.h>
long fesetenv(const fenv_t *f);
```

[引数]

f 浮動小数点環境を指すポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

設定する環境は fegetenv 関数または feholdexcept 関数にて設定した環境か、浮動小数点環境マクロと等しい環境を指定していください。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

feupdateenv

既出の例外を残したまま浮動小数点環境を設定します。

[指定形式]

```
#include <fenv.h>
long feupdateenv(const fenv_t *f);
```

[引数]

f 設定する浮動小数点環境を指すポインタ

[戻り値]

正常：0
異常：0 以外

[備考]

設定する浮動小数点環境は、fegetenv 関数または feholdexcept 関数の呼出しによって設定されたオブジェクトを指すか、又は浮動小数点環境マクロに等しいものにしてください。
本関数は、コンパイルオプション nofpu が選択されている場合は使用しないでください。
使用してもリターン値として異常を表す 0 以外を返します。

7.4.16 <inttypes.h>

整数型を拡張します。
以下は、すべて処理系定義です。

種別	定義名	説明
型 (マクロ)	imaxdiv_t	imaxdiv 関数の返す値の型です。
変数 (マクロ)	PRIdN PRIdLEASTN PRIdFASTN PRIdMAX PRIdPTR	
	PRIiN PRIiLEASTN PRIiFASTN PRIiMAX PRIiPTR	
	PRIoN PRIoLEASTN PRIoFASTN PRIoMAX PRIoPTR	
	PRIuN PRIuLEASTN PRIuFASTN PRIuMAX PRIuPTR	
	PRIxN PRIxLEASTN PRIxFASTN PRIxMAX PRIxPTR	
	PRIXN PRIXLEASTN PRIXFASTN PRIXMAX PRIXPTR	
	SCNdN SCNdLEASTN SCNdFASTN SCNdMAX SCNdPTR	
	SCNiN SCNiLEASTN SCNiFASTN SCNiMAX SCNiPTR	
	SCNoN SCNoLEASTN SCNoFASTN SCNoMAX SCNoPTR	

種別	定義名	説明
変数 (マクロ)	SCNuN SCNuLEASTN SCNuFASTN SCNuMAX SCNuPTR	
	SCNxN SCNxLEASTN SCNxFASTN SCNxMAX SCNxPTR	
関数	imaxabs	絶対値を計算する。
	imaxdiv	商、剰余を計算する。
	strtoimax / strtoumax	文字列最初の部分を intmax_t 型および uintmax_t 型表現に変換する以外は、strtol, strtoll, strtoul および strtoull 関数と等価。
	wcstoimax / wcstoumax	ワイド文字列最初の部分を intmax_t 型および uintmax_t 型表現に変換する以外は、wcstol, wcstoll, wcstoul および wcstoull 関数と等価。

imaxabs

絶対値を計算します。

[指定形式]

```
#include <inttypes.h>
intmax_t imaxabs(intmax_t a);
```

[引数]

a　絶対値を求める値

[戻り値]

a の絶対値

imaxdiv

除算 ("/") と剰余算 ("%") を一度に行います。

[指定形式]

```
#include <inttypes.h>
imaxdiv_t imaxdiv(intmax_t n, intmax_t d);
```

[引数]

n "/" 演算子または "%" 演算子の左辺
d "/" 演算子または "%" 演算子の右辺

[戻り値]

メンバとして quot (商) と rem (剰余) を持つ構造体 imaxdiv_t 型の値

[備考]

型 imaxdiv_t は次のように定義されています。

```
typedef struct {
    long long quot;
    long long rem;
} imaxdiv_t;
```

strtoimax / strtoumax

数を表現する文字列を intmax_t 型の整数に変換します。

[指定形式]

```
#include <inttypes.h>
intmax_t strtoimax(const char *nptr, char **endptr, long base);
uintmax_t strtoumax(const char *nptr, char **endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された intmax_t 型の整数値
異常：変換後の値がオーバフローの時：INTMAX_MAX, INTMAX_MIN または UINTMAX_MAX

[備考]

変換後の値がオーバフローをおこした時は、errno に ERANGE を設定します。
strtoimax 関数及び strtoumax 関数は文字列の最初の部分をそれぞれ intmax_t 型および
uintmax_t 型整数に変換するという点を除いて、strtol 関数、strtoll 関数、strtoul 関数及び strtoull 関数と等価とします。

wcstoimax / wcstoumax

数を表現する文字列を intmax_t 型または uintmax_t 型の整数に変換します。

[指定形式]

```
#include <stddef.h>
#include <inttypes.h>
intmax_t wcstoimax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
uintmax_t wcstoumax(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された intmax_t 型の整数値
異常：変換後の値がオーバフローの時：INTMAX_MAX, INTMAX_MIN または UINTMAX_MAX

[備考]

変換後の値がオーバフローをおこした時は、errno に ERANGE を設定します。
wcstrtoimax 関数及び wcstrtoumax 関数は文字列の最初の部分をそれぞれ intmax_t 型および uintmax_t 型整数に変換するという点を除いて、wcstol 関数、wcstoll 関数、wcstoul 関数及び wcstoull 関数と等価とします。

7.4.17 <iso646.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	and	&&
	and_eq	&=
	bitand	&
	bitor	|
	compl	~
	not	!
	not_eq	!=
	or	||
	or_eq	|=
	xor	^
	xor_eq	^=

7.4.18 <stdbool.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ (変数)	bool	_Bool に展開します。
マクロ (定数)	true	1 に展開します。
	false	0 に展開します。
	__bool_true_false_are_defined	1 に展開します。

7.4.19 <stdint.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	int_least8_t uint_least8_t int_least16_t uint_least16_t int_least32_t uint_least32_t int_least64_t uint_least64_t	8,16,32 および 64 ビットに対する、それぞれの符号あり / なし整数型を少なくとも格納できる大きさを持つ型です。
	int_fast8_t uint_fast8_t int_fast16_t uint_fast16_t int_fast32_t uint_fast32_t int_fast64_t uint_fast64_t	8,16,32 および 64 ビットに対する、それぞれの符号あり / なし整数型を最速で演算できる型です。
	intptr_t uintptr_t	void へのポインタを相互変換可能な符号あり / なし整数型です。
	intmax_t uintmax_t	すべての符号あり / なし整数型のすべての値を表現可能な符号あり / なし整数型です。
	int*N*_t uint*N*_t	N ビットの幅をもつ符号あり / なし整数型です。
	INT*N*_MIN INT*N*_MAX UINT*N*_MAX	幅指定符号あり整数型の最小値です。 幅指定符号あり整数型の最大値です。 幅指定符号なし整数型の最大値です。
	INT_LEAST*N*_MIN INT_LEAST*N*_MAX UINT_LEAST*N*_MAX	最小幅指定符号あり整数型の最小値です。 最小幅指定符号あり整数型の最大値です。 最小幅指定符号なし整数型の最大値です。
	INT_FAST*N*_MIN INT_FAST*N*_MAX UINT_FAST*N*_MAX	最速最小幅指定符号あり整数型の最小値です。 最速最小幅指定符号あり整数型の最大値です。 最速最小幅指定符号なし整数型の最大値です。
	INTPTR_MIN INTPTR_MAX UINTPTR_MAX	ポインタ保持可能な符号あり整数型の最小値です。 ポインタ保持可能な符号あり整数型の最大値です。 ポインタ保持可能な符号なし整数型の最大値です。
	INTMAX_MIN INTMAX_MAX UINTMAX_MAX	最大幅符号あり整数型の最小値です。 最大幅符号あり整数型の最大値です。 最大幅符号なし整数型の最大値です。
	PTRDIFF_MIN PTRDIFF_MAX	-2147483648 +2147483647
	SIG_ATOMIC_MIN SIG_ATOMIC_MAX	-2147483648 +2147483647
	SIZE_MAX	4294967295
	WCHAR_MIN WCHAR_MAX	0 65535U
	WINT_MIN WINT_MAX	0 65535U

種別	定義名	説明
関数 (マクロ)	INT*N*_C UINT*N*_C	Int_least*N*_t に対応する整数定数式に展開します。 uInt_least*N*_t に対応する整数定数式に展開します。
	INT_MAX_C UINT_MAX_C	intmax_t の整数定数式に展開します。 uintmax_t の整数定数式に展開します。

7.4.20 <tgmath.h>

以下は、すべてマクロ定義です。

型総称マクロ	<math.h> の関数	<complex.h> の関数
acos	acos	cacos
asin	asin	casin
atan	atan	catan
acosh	acosh	cacosh
asinh	asinh	casinh
atanh	atanh	catanh
cos	cos	ccos
sin	sin	csin
tan	tan	ctan
cosh	cosh	ccosh
sinh	sinh	csinh
tanh	tanh	ctanh
exp	exp	cexp
log	log	clog
pow	pow	cpow
sqrt	sqrt	csqrt
fabs	fabs	cfabs
atan2	atan2	-
cbrt	cbrt	-
ceil	ceil	-
copysign	copysign	-
erf	erf	-
erfc	erfc	-
exp2	exp2	-
expm1	expm1	-
fdim	fdim	-
floor	floor	-
fma	fma	-
fmax	fmax	-
fmin	fmin	-
fmod	fmod	-
frexp	frexp	-
hypot	hypot	-

型総称マクロ	<math.h> の関数	<complex.h> の関数
ilogb	ilogb	-
ldexp	ldexp	-
lgamma	lgamma	-
llrint	llrint	-
llround	llround	-
log10	log10	-
log1p	log1p	-
log2	log2	-
logb	logb	-
lrint	lrint	-
lround	lround	-
nearbyint	nearbyint	-
nextafter	nextafter	-
nexttoward	nexttoward	-
remainder	remainder	-
remquo	remquo	-
rint	rint	-
round	round	-
scalbn	scalbn	-
scalbln	scalbln	-
tgamma	tgamma	-
trunc	trunc	-
carg	-	carg
cimag	-	cimag
conj	-	conj
cproj	-	cproj
creal	-	creal

7.4.21 <wchar.h>

以下は、すべてマクロ定義です。

種別	定義名	説明
マクロ	mbstate_t	多バイト文字の並びとワイド文字の並びの間に必要な変換の状態を保持する型です。
	wint_t	拡張文字を保持する型です。
定数 (マクロ)	WEOF	ファイルの終わりを表します。
関数	fwprintf	出力形式を変換して、ストリームへ出力します。
	vfwprintf	可変個数の実引数並びを va_list で置き換えた fwprintf と等価です。
	swprintf	出力形式を変換してワイド文字の配列に書き込みます。
	vswprintf	可変個数の実引数並びを va_list で置き換えた swprintf と等価です。
	wprintf	与えられた実引数の前に stdout を実引数として付加した fwprintf と等価です。
	vwprintf	可変個数の実引数並びを va_list で置き換えた wprintf と等価です。
	fwscanf	ワイド文字列の制御に従ってストリームから入力して変換し、オブジェクトに代入します。
	vfwscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた fwscanf と等価です。
	swscanf	ワイド文字列の制御に従って変換し、オブジェクトに代入します。
	vswscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた swscanf と等価です。
	wscanf	与えられた実引数の前に stdin を実引数として付加した fwscanf と等価です。
	vwscanf <*-lang=c99*>	可変個数の実引数並びを va_list で置き換えた wscanf と等価です。
	fgetwc	wchar_t 型として取り込み wint_t 型に変換します。
	fgetws	ワイド文字の列を配列に格納します。
	fputwc	ワイド文字を書き込みます。
	fputws	ワイド文字列を書き込みます。
	fwide	入出力の単位を設定します。
	getwc	fgetwc と等価です。
	getwchar	実引数に stdin を指定した getwc と等価です。
	putwc	fputwc と等価です。
	putwchar	第２引数に stdout を指定した putwc と等価です。
	ungetwc	ワイド文字をストリームに戻します。
	wcstod / wcstof / wcstold	ワイド文字列の最初の部分を double, float および long double 型の表現に変換します。
	wcstol / wcstoll / wcstoul / wcstoull (wcstoll <*-lang=c99*>) (wcstoull <*-lang=c99*>)	ワイド文字列の最初の部分を long int, long long int, unsigned long int および unsigned long long int 型の表現に変換します。

種別	定義名	説明
関数	wcscpy	ワイド文字列をコピーします。
	wcsncpy	n 個以下のワイド文字をコピーします。
	wmemcpy	n ワイド文字をコピーします。
	wmemmove	n ワイド文字をコピーします。
	wcscat	ワイド文字列をコピーし、ワイド文字列の最後に付加します。
	wcsncat	n 個以下のワイド文字列をコピーし、ワイド文字列の最後に付加します。
	wcscmp	ワイド文字列同士を比較します。
	wcsncmp	n ワイド文字以下の配列を比較します。
	wmemcmp	n ワイド文字を比較します。
	wcschr	ワイド文字列の中でワイド文字を検索します。
	wcscspn	ワイド文字列の中で、ワイド文字列が含まれているかを検索します。
	wcspbrk	ワイド文字列の中で、ワイド文字列が含まれている最初の位置を検索します。
	wcsrchr	ワイド文字列の中でワイド文字が最後に現れる位置を検索します。
	wcsspn	ワイド文字列の中から、ワイド文字を含む先頭部分の最大の長さを計算します。
	wcsstr	ワイド文字列の中からワイド文字の並びをが最初に現れる位置を検索します。
	wcstok	ワイド文字列をワイド文字で区切られる字句の列に分割します。
	wmemchr	オブジェクトの先頭から n ワイド文字の中でワイド文字が最初に現れる位置を検索します。
	wcslen	ワイド文字列の長さを計算します。
	wmemset	n ワイド文字をコピーします。
	mbsinit	初期変換状態か判定します。
	mbrlen	多バイト文字を構成するバイト数を計算します。

fwprintf

書式に従って、ストリーム入出力用ファイルへデータを出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long fwprintf(FILE *restrict fp, const wchar_t *restrict control [, arg] ...);
```

[引数]

fp　ファイルポインタ
control　書式を示すワイド文字列へのポインタ
arg,...　書式に従って出力されるデータの並び

[戻り値]

正常：変換し出力したワイド文字列数
異常：負の値

[備考]

fwprintf 関数は fprintf 関数のワイド文字対応版です。

vfwprintf

可変個の引数リストを書式に従って、指定したストリーム入出力用ファイルに出力します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
long vfwprintf(FILE *restrict fp, const char *restrict control, va_list arg);
```

[引数]

fp　ファイルポインタ
control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

vfwprintf 関数は vfprintf 関数のワイド文字対応版です。

swprintf

データを書式に従って変換し、指定した領域へ出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long swprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control [, arg] ...);
```

[引数]

s　データを出力する記憶域へのポインタ
n　出力するワイド文字数
control　書式を示すワイド文字列へのポインタ
arg,...　書式に従って出力されるデータ

[戻り値]

正常：変換した文字数
異常：表現形式エラー又は n 個以上のワイド文字の書き込みが要求された場合は負の値

[備考]

swprintf 関数は sprintf 関数のワイド文字対応版です。

vswprintf

可変個の引数リストを書式に従って、指定した記憶域に出力します。

[指定形式]

```
#include <stdarg.h>
#include <wchar.h>
long vswprintf(wchar_t *restrict s, size_t n, const wchar_t *restrict control, va_list arg);
```

[引数]

s　データを出力する記憶域へのポインタ
n　出力するワイド文字数
control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換した文字数
異常：負の数

[備考]

vswprintf 関数は、vsprintf 関数のワイド文字対応版です。

wprintf

データを書式に従って変換し、標準出力ファイル (stdout) へ出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long wprintf(const wchar_t *restrict control [, arg] ...);
```

[引数]

control　書式を示す文字列へのポインタ
arg,...　書式に従って出力されるデータ

[戻り値]

正常：変換し出力したワイド文字数
異常：負の値

[備考]

wprintf 関数は printf 関数のワイド文字対応版です。

vwprintf

可変個の引数リストを書式に従って標準出力ファイル (stdout) に出力します。

[指定形式]

```
#include <stdarg.h>
#include <wchar.h>
long vwprintf(const wchar_t *restrict control, va_list arg);
```

[引数]

control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：変換し出力した文字数
異常：負の値

[備考]

vwprintf 関数は vprintf 関数のワイド文字対応版です。

fwscanf

ストリーム入出力ファイルからデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long fwscanf(FILE *restrict fp, const wchar_t *restrict control [, ptr] ...);
```

[引数]

fp　ファイルポインタ
control　書式を示すワイド文字列へのポインタ
ptr　入力したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

fwscanf 関数は、fscanf 関数のワイド文字対応版です。

vfwscanf

ストリーム入出力用ファイルからデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
long vfwscanf(FILE *restrict fp, const wchar_t *restrict control, va_list arg);
```

[引数]

fp　ファイルポインタ
control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

vfwscanf 関数は vfscanf 関数のワイド文字対応版です。

swscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long swscanf(const wchar_t *restrict s, const wchar_t *restrict control [, ptr] ...);
```

[引数]

s　入力するデータがある記憶域
control　書式を示すワイド文字列へのポインタ
ptr,...　入力変換したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：EOF

[備考]

swscanf 関数は sscanf 関数のワイド文字対応版です。

vswscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <wchar.h>
long vswscanf(const wchar_t *restrict s, const wchar_t *restrict control, va_list arg);
```

[引数]

s　入力するデータがある記憶域
control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：EOF

wscanf

標準入力ファイル (stdin) からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <wchar.h>
long wscanf(const wchar_t *control [, ptr] ...);
```

[引数]

control　書式を示すワイド文字列へのポインタ
ptr,...　入力変換したデータを格納する記憶域へのポインタ

[戻り値]

正常：入力変換に成功したデータの個数
異常：EOF

[備考]

wscanf 関数は scanf 関数のワイド文字対応版です。

vwscanf

指定した記憶域からデータを入力し、書式に従って変換します。

[指定形式]

```
#include <stdarg.h>
#include <wchar.h>
long vwscanf(const wchar_t *restrict control, va_list arg);
```

[引数]

control　書式を示すワイド文字列へのポインタ
arg　引数リスト

[戻り値]

正常：入力変換に成功したデータの個数
異常：入力データの変換を行う前に入力データが終了した時：EOF

[備考]

vwscanf 関数は、vscanf 関数をワイド文字列の書式を使えるようにした関数です。

fgetwc

ストリーム入出力用ファイルから 1 つのワイド文字を入力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wint_t fgetwc(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：EOF
　　ファイルの終了でない時：入力したワイド文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してのエラー指示子が設定されます。
fgetwc 関数は fgetc 関数をワイド文字が入力できるようにした関数です。

fgetws

ストリーム入出力用ファイルからワイド文字列を入力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wchar_t *fgetws(wchar_t *restrict s, long n, FILE *fp);
```

[引数]

s　ワイド文字列を入力する記憶域へのポインタ
n　ワイド文字列を入力する記憶域のバイト数
fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：NULL
　　ファイルの終了でない時：s
異常：NULL

[備考]

fgetws 関数は fgets 関数をワイド文字列が入力できるように対応した関数です。

fputwc

ストリーム入出力用ファイルへ 1 つのワイド文字を出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wint_t fputwc(wchar_t c, FILE *fp);
```

[引数]

c　出力する文字
fp　ファイルポインタ

[戻り値]

正常：出力したワイド文字
異常：EOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
fputwc 関数は fputc 関数のワイド文字対応版です。

fputws

ストリーム入出力用ファイルへワイド文字列を出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long fputws(const wchar_t *restrict s, FILE *restrict fp);
```

[引数]

s　出力するワイド文字列へのポインタ
fp　ファイルポインタ

[戻り値]

正常：0
異常：EOF

[備考]

fputws 関数は fputs 関数のワイド文字対応版です。

fwide

ファイルへの入力単位を設定します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long fwide(FILE *fp, long mode);
```

[引数]

fp　ファイルポインタ
mode　入力単位を表す値

[戻り値]

ワイド文字単位が設定された場合は 0 より大きい値
バイト単位の場合は 0 より小さい値
入出力単位をもたない場合は 0

[備考]

fwide 関数はストリーム入出力単位が既に決定されている場合、それを変更しません。

getwc

ストリーム入出力用ファイルから 1 つのワイド文字を入力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
long getwc(FILE *fp);
```

[引数]

fp　ファイルポインタ

[戻り値]

正常：ファイルの終了の時：WEOF
　　ファイルの終了でない時：入力した文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。
getwc 関数は fgetwc と等価ですが、マクロとして実装されているため、fp を 2 回以上評価することがあります。したがって、fp は副作用を伴わない式にしてください。

getwchar

標準入力ファイル (stdin) から、1 つのワイド文字を入力します。

[指定形式]

```
#include <wchar.h>
long getwchar(void);
```

[引数]

―

[戻り値]

正常：ファイルの終了の時：WEOF
　　ファイルの終了でない時：入力したワイド文字
異常：EOF

[備考]

読み込みエラーが発生した時、そのファイルに対してエラー指示子が設定されます。
getwchar 関数は getchar 関数のワイド文字対応版です。

putwc

ストリーム入出力用ファイルへ 1 つのワイド文字を出力します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wint_t putwc(wchar_t c, FILE *fp);
```

[引数]

c　出力するワイド文字
fp　ファイルポインタ

[戻り値]

正常：出力したワイド文字
異常：WEOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
putwc 関数は fputwc と等価ですが、マクロとして実装されているため、fp を 2 回以上評価することがあります。したがって、fp は副作用を伴わない式にしてください。

putwchar

標準出力ファイル (stdout) へ 1 つのワイド文字を出力します。

[指定形式]

```
#include <wchar.h>
wint_t putwchar(wchar_t c);
```

[引数]

c　出力するワイド文字

[戻り値]

正常：出力したワイド文字
異常：WEOF

[備考]

書き出しエラーが発生した時は、そのファイルに対してエラー指示子が設定されます。
putwchar 関数は putchar 関数のワイド文字対応版です。

ungetwc

ストリーム入出力用ファイルへ 1 つのワイド文字を戻します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc(wint_t c, FILE *fp);
```

[引数]

c　戻すワイド文字
fp　ファイルポインタ

[戻り値]

正常：戻したワイド文字
異常：WEOF

[備考]

ungetwc 関数は、ungetc 関数のワイド文字対応版です。

wcstod / wcstof / wcstold

ワイド文字列の最初の部分を所定の型の浮動小数点値に変換します。

[指定形式]

```
#include <wchar.h>
double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr, wchar_t **restrict endptr);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　浮動小数点値を構成していない最初の文字へのポインタを格納する記憶域へのポインタ

[戻り値]

正常：nptr が指している文字列が浮動小数点型を構成しない文字で始まっている時：0
　　nptr が指している文字列が浮動小数点型を構成する文字で始まっている時：変換された型の浮動小数点値
異常：変換後の値がオーバフローの時：変換する文字列の符号と同符号をもつ HUGE_VAL, HUGE_VALF, HUGE_VALL
　　変換後の値がアンダフローの時：0

[備考]

変換後の値がオーバフロー / アンダフローをおこした時は errno を設定します。
wcstod 関数群は strtod 関数群のワイド文字対応版です。

wcstol / wcstoll / wcstoul / wcstoull

ワイド文字列の最初の部分を所定の型の整数値に変換します。

[指定形式]

```
#include <wchar.h>
long int wcstol(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
long long int wcstoll(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
unsigned long int wcstoul(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
unsigned long long int wcstoull(const wchar_t * restrict nptr, wchar_t ** restrict endptr, long base);
```

[引数]

nptr　変換する数を表現する文字列へのポインタ
endptr　整数を構成しない最初の文字へのポインタを格納する記憶域へのポインタ
base　変換の基数 (0 又は 2 ～ 36)

[戻り値]

正常：nptr が指している文字列が整数を構成しない文字で始まっている時：0
　　nptr が指している文字列が整数を構成する文字で始まっている時：変換された型の整数値
異常：変換後の値がオーバフローの時：変換する文字列の符号に従って LONG_MIN、LONG_MAX、LLONG_MIN、LLONG_MAX、ULONG_MAX 又は ULLONG_MAX

[備考]

変換後の値がオーバフローをおこした時は、errno を設定します。
wcstol 関数群は strtol 関数群のワイド文字対応版です。

wcscpy

複写元のワイド文字列の内容を、複写先の記憶域にヌル文字も含めて複写します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcscpy(wchar_t * restrict s1, const wchar_t * restrict s2);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の文字列へのポインタ

[戻り値]

s1 の値

[備考]

wcscpy 関数群は strcpy 関数群のワイド文字対応版です。

wcsncpy

複写元のワイド文字列を指定された文字数分、複写先の記憶域に複写します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcsncpy(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の文字列へのポインタ
n　複写する文字数

[戻り値]

s1 の値

[備考]

wcsnpy 関数は strncpy 関数のワイド文字対応版です。

wmemcpy

複写元の記憶域の内容を、指定した大きさ分、複写先の記憶域に複写します。

[指定形式]

```
#include <wchar.h>
wchar_t *wmemcpy(wchar_t *restrict s1, const wchar_t *restrict s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の記憶域へのポインタ
n　複写する文字数

[戻り値]

s1 の値

[備考]

wmemcpy 関数は memcpy 関数のワイド文字対応版です。

wmemmove

複写元の記憶域の内容を指定した大きさ分、複写先の記憶域に複写します。また、複写元と複写先の記憶域が、重なっている部分があっても、複写元の重なっている部分を上書きする前に複写するので正しく複写されます。

[指定形式]

```
#include <wchar.h>
wchar_t *wmemmove(wchar_t *s1, const wchar_t *s2, size_t n);
```

[引数]

s1　複写先の記憶域へのポインタ
s2　複写元の記憶域へのポインタ
n　複写する文字数

[戻り値]

s1 の値

[備考]

wmemmove 関数は memmove 関数のワイド文字対応版です。

wcscat

文字列の後に、文字列を連結します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcscat(wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　連結される文字列へのポインタ
s2　連結する文字列へのポインタ

[戻り値]

s1 の値

[備考]

wcscat 関数は strcat 関数のワイド文字対応版です。

wcsncat

文字列に文字列を指定した文字数分連結します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcsncat(wchar_t * restrict s1, const wchar_t * restrict s2, size_t n);
```

[引数]

s1　連結される文字列へのポインタ
s2　連結する文字列へのポインタ
n　連結する文字数

[戻り値]

s1 の値

[備考]

wcsncat 関数は strncat 関数のワイド文字対応版です。

wcscmp

指定された 2 つの文字列の内容を比較します。

[指定形式]

```
#include <wchar.h>
long wcscmp(const wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ

[戻り値]

s1 で指された文字列 >s2 で指された文字列の時：正の値
s1 で指された文字列 ==s2 で指された文字列の時：0
s1 で指された文字列 <s2 で指された文字列の時：負の値

[備考]

wcscmp 関数は strcmp 関数のワイド文字対応版です。

wcsncmp

指定された 2 つの文字列を指定された文字分まで比較します。

[指定形式]

```
#include <wchar.h>
long wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t n);
```

[引数]

s1　比較される文字列へのポインタ
s2　比較する文字列へのポインタ
n　比較する文字数の最大値

[戻り値]

s1 で指された文字列 >s2 で指された文字列の時：正の値
s1 で指された文字列 ==s2 で指された文字列の時：0
s1 で指された文字列 <s2 で指された文字列の時：負の値

[備考]

wcsncmp 関数は strncmp 関数のワイド文字対応版です。

wmemcmp

指定された 2 つの記憶域の内容を比較します。

[指定形式]

```
#include <wchar.h>
long wmemcmp(const wchar_t * s1, const wchar_t * s2, size_t n);
```

[引数]

s1　比較される記憶域へのポインタ
s2　比較する記憶域へのポインタ
n　比較する記憶域の文字数

[戻り値]

s1 で指された記憶域 >s2 で指された記憶域の時：正の値
s1 で指された記憶域 ==s2 で指された記憶域の時：0
s1 で指された記憶域 <s2 で指された記憶域の時：負の値

[備考]

wmemcmp 関数は memcmp 関数のワイド文字対応版です。

wcschr

指定された文字列において、指定された文字が最初に現われる位置を検索します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcschr(const wchar_t *s, wchar_t c);
```

[引数]

s　検索を行う文字列へのポインタ
c　検索する文字

[戻り値]

検索の結果見つかった時：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

wcschr 関数は strchr 関数のワイド文字対応版です。

wcscspn

指定された文字列を先頭から調べ、別に指定した文字列中の文字以外の文字が先頭から何文字続くか求めます。

[指定形式]

```
#include <wchar.h>
size_t wcscspn(const wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　調べられる文字列へのポインタ
s2　s1 を調べるための文字列へのポインタ

[戻り値]

s2 が指す文字列を構成する文字以外の文字が構成される文字列 s1 の先頭からの長さ

[備考]

wcscspn 関数は strcspn 関数のワイド文字対応版です。

wcspbrk

指定された文字列内において、別に指定された文字列中の文字が最初に現われる位置を検索します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcspbrk(const wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　検索を行う文字列へのポインタ
s2　s1 内で検索する文字を示す文字列へのポインタ

[戻り値]

検索の結果見つかった時：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

wcspbrk 関数は strpbrk 関数のワイド文字対応版です。

wcsrchr

指定された文字列において、指定された文字が最後に現われる位置を検索します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcsrchr(const wchar_t *s, wchar_t c);
```

[引数]

s　検索を行う文字列へのポインタ
c　検索する文字

[戻り値]

検索の結果見つかった時：見つかった文字へのポインタ
検索の結果見つからなかった時：NULL

wcsspn

指定された文字列を先頭から調べ、別に指定した文字列中の文字が先頭から何文字続くかを求めます。

[指定形式]

```
#include <wchar.h>
size_t wcsspn(const wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　調べられる文字列へのポインタ
s2　s1 を調べるための文字列へのポインタ

[戻り値]

s1 の先頭から、s2 で指定した文字が続いている文字数

[備考]

wcsspn 関数は strspn 関数のワイド文字対応版です。

wcsstr

指定された文字列において、別に指定した文字列が最初に現われる位置を検索します。

[指定形式]

```
#include <wchar.h>
wchar_t *wcsstr(const wchar_t *s1, const wchar_t *s2);
```

[引数]

s1　検索を行う文字列へのポインタ
s2　検索する文字列へのポインタ

[戻り値]

検索の結果見つかったとき：見つけられた文字へのポインタ
検索の結果見つからなかったとき：NULL

wcstok

指定した文字列をいくつかの字句に切り分けます。

[指定形式]

```
#include <wchar.h>
wchar_t* wcstok(wchar_t * restrict s1, const wchar_t * restrict s2, wchar_t ** restrict ptr);
```

[引数]

s1　いくつかの字句に切り分ける文字列へのポインタ
s2　文字列を切り分けるための文字からなる文字列へのポインタ
ptr　次の関数呼び出し時に検索を始める文字列へのポインタ

[戻り値]

字句に切り分けられた時：切り分けた字句の先頭へのポインタ
字句に切り分けられなかった時：NULL

[備考]

wcstok 関数は strtok 関数のワイド文字対応版です。
同じ文字列に対して 2 回目以降の検索をする場合は s1 に NULL を設定し、ptr には
前回の同文字列に対する関数呼び出しで取得した値を設定してください。

wmemchr

指定された記憶域において、指定された文字が最初に現われる位置を検索します。

[指定形式]

```
#include <wchar.h>
wchar_t *wmemchr(const wchar_t *s, wchar_t c, size_t n);
```

[引数]

s　検索を行う記憶域へのポインタ
c　検索する文字
n　検索を行う文字数

[戻り値]

検索の結果見つかった時：見つけられた文字へのポインタ
検索の結果見つからなかった時：NULL

[備考]

wmemchr 関数は memchr 関数のワイド文字対応版です。

wcslen

終端ナルワイド文字を除くワイド文字列の長さを計算します。

[指定形式]

```
#include <wchar.h>
size_t wcslen(const wchar_t *s);
```

[引数]

s 長さをワイド文字列へのポインタ

[戻り値]

ワイド文字列の文字数

[備考]

wcslen 関数は strlen 関数のワイド文字対応版です。

wmemset

指定された記憶域の先頭から、指定された文字を指定された文字数分設定します。

[指定形式]

```
#include <stdio.h>
#include <wchar.h>
wchar_t *wmemset(wchar_t *s, wchar_t c, size_t n);
```

[引数]

s 文字が設定される記憶域へのポインタ
c 設定する文字
n 設定する文字数

[戻り値]

s の値

[備考]

wmemset 関数は memset 関数のワイド文字対応版です。

mbsinit

指定された mbstate_t オブジェクトが初期変換状態かどうか判定します。

[指定形式]

```
#include <wchar.h>
long mbsinit(const mbstate_t *ps);
```

[引数]

ps　mbstate_t オブジェクトへのポインタ

[戻り値]

初期化状態の場合 0 以外の値
それ以外の状態の場合は 0

mbrlen

指定された多バイト文字のバイト数を取得します。

[指定形式]

```
#include <wchar.h>
size_t mbrlen(const char * restrict s, size_t n, mbstate_t *restrict ps);
```

[引数]

s　多バイト文字列へのポインタ
n　認識する多バイト文字の最大バイト数
ps　mbstate_t オブジェクトへのポインタ

[戻り値]

0：n 個以下のバイトによってナルワイド文字を認識した場合
1 以上 n 以下：n 個以下のバイトによって多バイト文字を認識した場合
(size_t)(-2)：n 個のバイトだけでは完全な多バイト文字を認識できない場合
(size_t)(-1)：不正な多バイト文字の並びに遭遇した場合

7.5 EC++ ライブラリ関数

この節では、C++ プログラムから標準的に利用できる EC++ クラスライブラリの仕様について説明します。ここでは、クラスライブラリの種類と対応する標準インクルードファイルについて説明します。以降では、ライブラリの構成に従って各クラスライブラリの仕様について説明します。

- ライブラリの種類
  表 7.15 にクラスライブラリの種類と対応する標準インクルードファイルを示します。

表 7.15 クラスライブラリの種類と標準インクルードファイルの対応

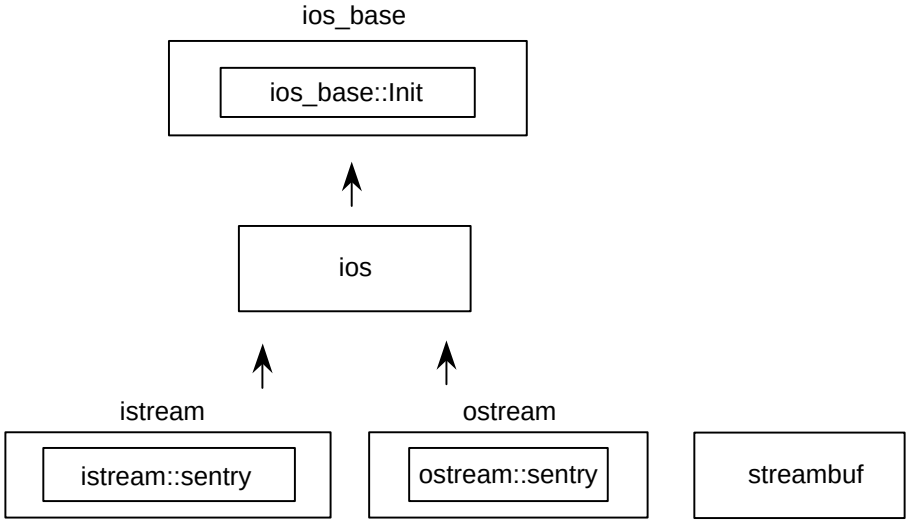
	ライブラリの種類	内容	標準 インクルードファイル
1	ストリーム入出力用クラスライブラリ	入出力操作を行うライブラリです。	<ios>,<streambuf>, <istream>,<ostream>, <iostream>,<iomanip>
2	メモリ操作用ライブラリ	メモリの確保・解放を行うライブラリです。	<new>
3	複素数計算用クラスライブラリ	複素数データ演算を行うライブラリです。	<complex>
4	文字列操作用クラスライブラリ	文字列操作を行うライブラリです。	<string>

7.5.1 ストリーム入出力用クラスライブラリ

ストリーム入出力用クラスライブラリに対応するヘッダファイルは以下の通りです。

- <ios>
  入出力用書式設定、入出力状態管理を行うデータメンバおよび関数メンバを定義します。
  ios クラスの他に、Init クラス、ios_base クラスを定義します。
- <streambuf>
  ストリームバッファに対する関数を定義します。
- <istream>
  入力ストリームからの入力関数を定義します。
- <ostream>
  出力ストリームへの出力関数を定義します。
- <iostream>
  入出力関数を定義します。
- <iomanip>
  引数を持つマニピュレータを定義します。

これらのクラスの派生関係は次のようになります。矢印は、派生クラスから基底クラスを参照していることを示します。なお、streambuf クラスには派生関係はありません。



ストリーム入出力用クラスライブラリで共通に使用される型名を示します。

種別	定義名	説明
型	streamoff	long 型で定義された型です。
	streamsize	size_t 型で定義された型です。
	int_type	int 型で定義された型です。
	pos_type	long 型で定義された型です。
	off_type	long 型で定義された型です。

(a) ios_base::Init クラス

種別	定義名	説明
変数	init_cnt	ストリーム入出力オブジェクト数をカウントする静的データメンバです。 低水準インタフェースで 0 に初期化する必要があります。
関数	Init()	コンストラクタです
	~Init()	デストラクタです。

ios_base::Init::Init()

クラス Init のコンストラクタです。
init_cnt をインクリメントします。

ios_base::Init::~Init()

クラス Init のデストラクタです。
init_cnt をデクリメントします。

(b) ios_base クラス

種別	定義名	説明
型	fmtflags	フォーマット制御情報を表す型です。
	iostate	ストリームバッファの入出力状態を表す型です。
	openmode	ファイルのオープンモードを表す型です。
	seekdir	ストリームバッファのシーク状態を表す型です。
変数	fmtfl	書式フラグです。
	wide	フィールド幅です。
	prec	出力時の精度 (小数点以下の桁数) です。
	fillch	詰め文字です。

種別	定義名	説明
関数	void _ec2p_init_base()	初期化します。
	void _ec2p_copy_base(ios_base&ios_base_dt)	ios_base_dt をコピーします。
	ios_base()	コンストラクタです。
	~ios_base()	デストラクタです。
	fmtflags flags() const	書式フラグ (fmtfl) を参照します。
	fmtflags flags(fmtflags fmtflg)	fmtflg&書式フラグ(fmtfl)を書式フラグ(fmtfl)に設定します。
	fmtflags setf(fmtflags fmtflg)	fmtflg を書式フラグ (fmtfl) に設定します。
	fmtflags setf(fmtflags fmtflg, fmtflags mask)	mask&fmtflg を書式フラグ (fmtfl) に設定します。
	void unsetf(fmtflags mask)	~mask& 書式フラグ (fmtfl) を書式フラグ (fmtfl) に設定します。
	char fill() const	詰め文字 (fillch) を参照します。
	char fill(char ch)	ch を詰め文字 (fillch) に設定します。
	int precision() const	精度 (prec) を参照します。
	streamsize precision(streamsize preci)	preci を精度 (prec) に設定します。
	streamsize width() const	フィールド幅 (wide) を参照します。
	streamsize width(streamsize wd)	wd をフィールド幅 (wide) に設定します。

ios_base::fmtflags

入出力に関するフォーマット制御情報を定義します。
fmtflags の各ビットマスクの定義は以下のようになります。
const ios_base::fmtflags ios_base::boolalpha=0x0000;
const ios_base::fmtflags ios_base::skipws=0x0001;
const ios_base::fmtflags ios_base::unitbuf=0x0002;
const ios_base::fmtflags ios_base::uppercase=0x0004;
const ios_base::fmtflags ios_base::showbase=0x0008;
const ios_base::fmtflags ios_base::showpoint=0x0010;
const ios_base::fmtflags ios_base::showpos=0x0020;
const ios_base::fmtflags ios_base::left=0x0040;
const ios_base::fmtflags ios_base::right=0x0080;
const ios_base::fmtflags ios_base::internal=0x0100;
const ios_base::fmtflags ios_base::adjustfield=0x01c0;
const ios_base::fmtflags ios_base::dec=0x0200;
const ios_base::fmtflags ios_base::oct=0x0400;
const ios_base::fmtflags ios_base::hex=0x0800;
const ios_base::fmtflags ios_base::basefield=0x0e00;
const ios_base::fmtflags ios_base::scientific=0x1000;
const ios_base::fmtflags ios_base::fixed=0x2000;
const ios_base::fmtflags ios_base::floatfield=0x3000;
const ios_base::fmtflags ios_base::_fmtmask=0x3fff;

ios_base::iostate

ストリームバッファの入出力状態を定義します。
iostate の各ビットマスクの定義は以下のようになります。
const ios_base::iostate ios_base::goodbit=0x0;
const ios_base::iostate ios_base::eofbit=0x1;
const ios_base::iostate ios_base::failbit=0x2;
const ios_base::iostate ios_base::badbit=0x4;
const ios_base::iostate ios_base::_statemask=0x7;

ios_base::openmode

ファイルのオープンモードを定義します。
openmode の各ビットマスクの定義は以下のようになります。
const ios_base::openmode ios_base::in=0x01;　入力用のファイルを open します。
const ios_base::openmode ios_base::out=0x02;　出力用のファイルを open します。
const ios_base::openmode ios_base::ate=0x04;　オープン後一度だけ eof に seek します。
const ios_base::openmode ios_base::app=0x08;　書き込む度に eof に seek します。
const ios_base::openmode ios_base::trunc=0x10;　ファイルを上書きモードで open します。
const ios_base::openmode ios_base::binary=0x20;　ファイルをバイナリモードで open します。

ios_base::seekdir

ストリームバッファのシーク状態を定義します。
引き続き入力または出力を行うためのストリーム内の位置を決定します。
seekdir の各ビットマスクの定義は以下のようになります。
const ios_base::seekdir ios_base::beg=0x0;
const ios_base::seekdir ios_base::cur=0x1;
const ios_base::seekdir ios_base::end=0x2;

void ios_base::_ec2p_init_base()

以下の値で初期設定します。
fmtfl = skipws | dec;
wide = 0;
prec = 6;
fillch = ' ';

void ios_base::_ec2p_copy_base(ios_base& ios_base_dt)

ios_base_dt をコピーします。

ios_base::ios_base()

クラス ios_base のコンストラクタです。
Init::Init() を呼び出します。

ios_base::~ios_base()

クラス ios_base のデストラクタです。

ios_base::fmtflags ios_base::flags() const

書式フラグ (fmtfl) を参照します。
リターン値は、書式フラグ (fmtfl) です。

ios_base::fmtflags ios_base::flags(fmtflags fmtflg)

fmtflg& 書式フラグ (fmtfl) を書式フラグ (fmtfl) に設定します。
リターン値は、設定前の書式フラグ (fmtfl) です。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg)

fmtflg を書式フラグ (fmtfl) に設定します。
リターン値は、設定前の書式フラグ (fmtfl) です。

ios_base::fmtflags ios_base::setf(fmtflags fmtflg, fmtflags mask)

mask&fmtflg の値を書式フラグ (fmtfl) に設定します。
リターン値は、設定前の書式フラグ (fmtfl) です。

void ios_base::unsetf(fmtflags mask)

~mask& 書式フラグ (fmtfl) を書式フラグ (fmtfl) に設定します。

char ios_base::fill() const

詰め文字 (fillch) を参照します。
リターン値は、詰め文字 (fillch) です。

char ios_base::fill(char ch)

ch を詰め文字として設定します。
リターン値は、設定前の詰め文字 (fillch) です。

int ios_base::precision() const

精度 (prec) を参照します。
リターン値は、精度 (prec) です。

streamsize ios_base::precision(streamsize preci)

preci を精度 (prec) に設定します。
リターン値は、設定前の精度 (prec) です。

streamsize ios_base::width() const

フィールド幅 (wide) を参照します。
リターン値は、フィールド幅 (wide) です。

streamsize ios_base::width(streamsize wd)

wd をフィールド幅 (wide) に設定します。
リターン値は、設定前のフィールド幅 (wide) です。

(c) ios クラス

種別	定義名	説明
変数	sb	streambuf オブジェクトへのポインタです。
	tiestr	ostream オブジェクトへのポインタです。
	state	streambuf への状態フラグです。
関数	ios()	コンストラクタです。
	ios(streambuf* sbptr)	
	void init(streambuf* sbptr)	初期設定を行います。
	virtual ~ios()	デストラクタです。
	operator void*() const	エラー有無 (!state&(badbit | failbit)) を判定します。
	bool operator!() const	エラー有無 (state&(badbit | failbit)) を判定します。
	iostate rdstate() const	状態フラグ (state) を参照します。
	void clear(iostate st = goodbit)	指定された状態 (st) を除いて状態フラグ (state) をクリアします。
	void setstate(iostate st)	st を状態フラグ (state) に設定します。
	bool good() const	エラー有無 (state==goodbit) を判定します。
	bool eof() const	入力ストリームの最後かどうか (state&eofbit) を判定します。
	bool bad() const	エラー有無 (state&badbit) を判定します。
	bool fail() const	入力テキストが要求パターンと不一致であるかどうか (state&(badbit | failbit)) 判定します。
	ostream* tie() const	ostream オブジェクトへのポインタ (tiestr) を参照します。
	ostream* tie(ostream* tstrptr)	tstrptr を ostream オブジェクトへのポインタ (tiestr) に設定します。
	streambuf* rdbuf() const	streambuf オブジェクトへのポインタ (sb) を参照します。
	streambuf* rdbuf(streambuf* sbptr)	sbptr を streambuf オブジェクトへのポインタ (sb) に設定します。
	ios& copyfmt(const ios& rhs)	rhs の状態フラグ (state) をコピーします。

ios::ios()

クラス ios のコンストラクタです。
init(0) を呼び出し、初期値をそのメンバオブジェクトに設定します。

ios::ios(streambuf* sbptr)

クラス ios のコンストラクタです。
init(sbptr) を呼び出し、初期値をそのメンバオブジェクトに設定します。

```
void ios::init(streambuf* sbptr)
```

sbptr を sb に設定します。
state、tiestr を 0 に設定します。

```
virtual ios::~ios()
```

クラス ios のデストラクタです。

```
ios::operator void*() const
```

エラー有無 (!state&(badbit | failbit)) を判定します。
リターン値は次のとおりです。
　エラー有の場合：false
　エラー無の場合：true

```
bool ios::operator!() const
```

エラー有無 (state&(badbit | failbit)) を判定します。
リターン値は次のとおりです。
　エラー有の場合：true
　エラー無の場合：false

```
iostate ios::rdstate() const
```

状態フラグ (state) を参照します。
リターン値は、状態フラグ (state) です。

```
void ios::clear(iostate st = goodbit)
```

指定された状態 (st) を除いて状態フラグ (state) をクリアします。
streambuf オブジェクトへのポインタ (sb) が 0 のときは、状態フラグ (state) に badbit を設定します。

```
void ios::setstate(iostate st)
```

st を状態フラグ (state) に設定します。

```
bool ios::good() const
```

エラー有無 (state==goodbit) を判定します。
リターン値は次のとおりです。
　エラー有の場合：false
　エラー無の場合：true

```
bool ios::eof() const
```

入力ストリームの最後かどうか (state&eofbit) を判定します。
リターン値は次のとおりです。
　入力ストリームの最後の場合：true
　入力ストリームの最後以外の場合：false

bool ios::bad() const

エラー有無 (state&badbit) を判定します。
リターン値は次のとおりです。
　エラー有の場合：true
　エラー無の場合：false

bool ios::fail() const

入力テキストが要求パターンと不一致であるかどうか (state&(badbit | failbit)) を判定します。
リターン値は次のとおりです。
　不一致の場合：true
　一致の場合：false

ostream* ios::tie() const

ostream オブジェクトへのポインタ (tiestr) を参照します。
リターン値は、ostream オブジェクトへのポインタ (tiestr) です。

ostream* ios::tie(ostream* tstrptr)

tstrptr を ostream オブジェクトへのポインタ (tiestr) に設定します。
リターン値は、設定前の ostream オブジェクトへのポインタ (tiestr) です。

streambuf* ios::rdbuf() const

streambuf オブジェクトへのポインタ (sb) を参照します。
リターン値は、streambuf オブジェクトへのポインタ (sb) です。

streambuf* ios::rdbuf(streambuf* sbptr)

sbptr を streambuf オブジェクトへのポインタ (sb) に設定します。
リターン値は、設定前の streambuf オブジェクトへのポインタ (sb) です。

ios& ios::copyfmt(const ios& rhs)

rhs の状態フラグ (state) をコピーします。
リターン値は *this です。

(d) ios クラスマニピュレータ

種別	定義名	説明
関数	ios_base& showbase(ios_base& str)	基数表示接頭辞モードに設定します。
	ios_base& noshowbase(ios_base& str)	基数表示接頭辞モードをクリアします。
	ios_base& showpoint (ios_base& str)	小数点生成モードに設定します。
	ios_base& noshowpoint (ios_base& str)	小数点生成モードをクリアします。
	ios_base& showpos(ios_base& str)	+ 記号生成モードに設定します。
	ios_base& noshowpos(ios_base& str)	+ 記号生成モードをクリアします。
	ios_base& skipws(ios_base& str)	空白読み飛ばしモードに設定します。
	ios_base& noskipws(ios_base& str)	空白読み飛ばしモードをクリアします。
	ios_base& uppercase(ios_base& str)	大文字変換モードに設定します。
	ios_base& nouppercase(ios_base& str)	大文字変換モードをクリアします。
	ios_base& internal(ios_base& str)	内部補充モードに設定します。
	ios_base& left(ios_base& str)	左側補充モードに設定します。
	ios_base& right(ios_base& str)	右側補充モードに設定します。
	ios_base& dec(ios_base& str)	10 進モードに設定します。
	ios_base& hex(ios_base& str)	16 進モードに設定します。
	ios_base& oct(ios_base& str)	8 進モードに設定します。
	ios_base& fixed(ios_base& str)	固定小数点モードに設定します。
	ios_base& scientific(ios_base& str)	科学表記法モードに設定します。

ios_base& showbase(ios_base& str)

データのはじめに基数を表示させるモードに設定します。
16 進数のときは、0x を行の先頭に付加します。10 進数のときは、そのまま出力します。
8 進数のときは、0 を行の先頭に付加します。
リターン値は str です。

ios_base& noshowbase(ios_base& str)

データのはじめに基数を表示させるモードをクリアします。
リターン値は str です。

ios_base& showpoint(ios_base& str)

小数点を出力するモードに設定します。
精度の指定がない場合、小数点以下 6 桁で表示します。
リターン値は str です。

ios_base& noshowpoint(ios_base& str)

小数点を出力するモードをクリアします。
リターン値は str です。

ios_base& showpos(ios_base& str)

+ 記号生成出力モード (正の数に対して + の符号を付加) に設定します。
リターン値は str です。

ios_base& noshowpos(ios_base& str)

+ 記号生成出力モードをクリアします。
リターン値は str です。

ios_base& skipws(ios_base& str)

空白読み飛ばし入力モード (連続する空白をスキップ) に設定します。
リターン値は str です。

ios_base& noskipws(ios_base& str)

空白読み飛ばし入力モードをクリアします。
リターン値は str です。

ios_base& uppercase(ios_base& str)

大文字変換出力モードに設定します。
16 進の基数表現が大文字の 0X になり、数値自体も大文字になります。
浮動小数点の指数表現も大文字の E になります。
リターン値は str です。

ios_base& nouppercase(ios_base& str)

大文字変換出力モードをクリアします。
リターン値は str です。

ios_base& internal(ios_base& str)

フィールド幅 (wide) の範囲で出力時に
　符号、基数
　詰め文字 (fill)
　数値
の順で出力します。
リターン値は str です。

ios_base& left(ios_base& str)

フィールド幅 (wide) の範囲で出力時に左詰めします。
リターン値は str です。

ios_base& right(ios_base& str)

フィールド幅 (wide) の範囲で出力時に右詰めします。
リターン値は str です。

ios_base& dec(ios_base& str)

変換基数を 10 進モードに設定します。
リターン値は str です。

ios_base& hex(ios_base& str)

変換基数を 16 進モードに設定します。
リターン値は str です。

ios_base& oct(ios_base& str)

変換基数を 8 進モードに設定します。
リターン値は str です。

ios_base& fixed(ios_base& str)

固定小数点出力モードに設定します。
リターン値は str です。

ios_base& scientific(ios_base& str)

科学表記法出力モード (指数表記) に設定します。
リターン値は str です。

(e) streambuf クラス

種別	定義名	説明
定数	eof	ファイル終了を示します。
変数	_B_cnt_ptr	バッファの有効データ長へのポインタです。
	B_beg_ptr	バッファのベースポインタへのポインタです。
	_B_len_ptr	バッファの長さへのポインタです。
	B_next_ptr	バッファの次の読み出し位置へのポインタです。
	B_end_ptr	バッファの終端位置へのポインタです。
	B_beg_pptr	制御バッファの先頭位置へのポインタです。
	B_next_pptr	バッファの次の読み出し位置へのポインタです。
	C_flg_ptr	ファイルの入出力制御フラグへのポインタです。

種別	定義名	説明
関数	char* _ec2p_getflag() const	ファイル入出力制御フラグのポインタを参照します。
	char*& _ec2p_gnptr()	バッファの次の読み出し位置へのポインタを参照します。
	char*& _ec2p_pnptr()	バッファの次の書き込み位置へのポインタを参照します。
	void _ec2p_bcntplus()	バッファの有効データ長をインクリメントします。
	void _ec2p_bcntminus()	バッファの有効データ長をデクリメントします。
	void _ec2p_setbPtr(char** begptr, char** curptr, long* cntptr, long* lenptr, char* flgptr)	streambuf のポインタを設定します。
	streambuf()	コンストラクタです。
	virtual ~streambuf()	デストラクタです。
	streambuf* pubsetbuf(char* s, streamsize n)	ストリーム入出力用のバッファを確保します。この関数では setbuf(s,n)*1 を呼び出します。
	pos_type pubseekoff(off_type off, ios _base::seekdir way, ios_base::openmode which = ios_base::in | ios_base::out)	wayで指定された方法で入出力ストリームの読み書き位置を移動させます。この関数では seekoff(off,way,which)*1 を呼び出します。
	pos_type pubseekpos(pos_type sp, ios_base::openmode which = ios_base::in | ios_base::out)	ストリームの先頭から現在の位置までのオフセットを求めます。この関数では seekpos(sp,which)*1 を呼び出します。
	int pubsync()	出力ストリームをフラッシュします。この関数では sync()*1 を呼び出します。
	streamsize in_avail()	入力ストリームの最後尾から現在位置までのオフセットを求めます。
	int_type snextc()	次の一文字を読み込みます。
	int_type sbumpc()	一文字読み込みポインタを次に設定します。
	int_type sgetc()	一文字読み込みます。
	int sgetn(char* s, streamsize n)	s の指す記憶領域に n 個の文字を設定します。
	int_type sputbackc(char c)	読み込み位置をプットバックします。
	int sungetc()	読み込み位置をプットバックします。
	int sputc(char c)	文字 c を挿入します。
	int_type sputn(const char* s, streamsize n)	s の指す n 個の文字を挿入します。
	char* eback() const	入力ストリームの先頭ポインタを求めます。
	char* gptr() const	入力ストリームの次ポインタを求めます。
	char* egptr() const	入力ストリームの最後尾ポインタを求めます。

種別	定義名	説明
関数	void gbump(int n)	入力ストリームの次ポインタを n 進めます。
	void setg(char* gbeg, char* gnext, char* gend)	入力ストリームの各ポインタを代入します。
	char* pbase() const	出力ストリームの先頭ポインタを求めます。
	char* pptr() const	出力ストリームの次ポインタを求めます。
	char* epptr() const	出力ストリームの最後尾ポインタを求めます。
	void pbump(int n)	出力ストリームの次ポインタを n 進めます。
	void setp(char* pbeg, char* pend)	出力ストリームの各ポインタを設定します。
	virtual streambuf* setbuf(char* s, streamsize n)*1	派生する各クラスごとに、個別に定義する演算を実行します。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))*1	ストリーム位置を変更します。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out)) *1	ストリーム位置を変更します。
	virtual int sync()*1	出力ストリームをフラッシュします。
	virtual int showmanyc()*1	入力ストリームの有効な文字数を求めます。
	virtual streamsize xsgetn(char* s, streamsize n)	s の指す記憶領域に n 個の文字を設定します。
	virtual int_type underflow()*1	ストリーム位置を動かさずに一文字読み込みます。
	virtual int_type uflow()*1	次ポインタの一文字を読み込みます。
	virtual int_type pbackfail(int_type c = eof)*1	c によって示される文字をプットバックします。
	virtual streamsize xsputn(const char* s, streamsize n)	s の指す n 個の文字を挿入します。
	virtual int_type overflow(int_type c = eof)*1	c を出力ストリームに挿入します。

注 1. このクラスでは処理を定義していません。

```
char* streambuf::_ec2p_getflag() const
```

ファイル入出力制御フラグのポインタを参照します。

```
char*& streambuf::_ec2p_gnptr()
```

バッファの次の読み出し位置へのポインタを参照します。

```
char*& streambuf::_ec2p_pnptr()
```

バッファの次の書き込み位置へのポインタを参照します。

```
void streambuf::_ec2p_bcntplus()()
```

バッファの有効データ長をインクリメントします。

```
void streambuf::_ec2p_bcntminus()
```

バッファの有効データ長をデクリメントします。

```
void _ec2p_setbPtr(char** begptr, char** curptr, long* cntptr, long* lenptr, char* flgptr)
```

streambuf のポインタを設定します。

```
streambuf::streambuf()
```

コンストラクタです。
以下の値で初期化します。
_B_cnt_ptr=B_beg_ptr=B_next_ptr=B_end_ptr=C_flg_ptr=_B_len_ptr=0
B_beg_pptr=&B_beg_ptr
B_next_pptr=&B_next_ptr

```
virtual streambuf::~streambuf()
```

デストラクタです。

```
streambuf* streambuf::pubsetbuf(char* s, streamsize n)
```

ストリーム入出力用のバッファを確保します。
この関数では setbuf(s,n) を呼び出します。
リターン値は、*this です。

```
pos_type streambuf::pubseekoff(off_type off, ios _base::seekdir way, ios_base::openmode which =
(ios_base::openmode)(ios_base::in | ios_base::out))
```

way で指定された方法で入出力ストリームの読み書き位置を移動させます。
この関数では seekoff(off,way,which) を呼び出します。
リターン値は、新たに設定されたストリームの位置です。

```
pos_type streambuf::pubseekpos(pos_type sp, ios_base::openmode which =
(ios_base::openmode)(ios_base::in | ios_base::out))
```

ストリームの先頭から現在の位置までのオフセットを求めます。
現在のストリームポインタから sp だけ移動します。
この関数では seekpos(sp,which) を呼び出します。
リターン値は、先頭からのオフセットです。

```
int streambuf::pubsync()
```

出力ストリームをフラッシュします。
この関数では sync() を呼び出します。
リターン値は 0 です。

streamsize streambuf::in_avail()

入力ストリームの最後尾から現在位置までのオフセットを求めます。
リターン値は次のとおりです。
　読み込み位置が有効の場合：最後尾から現在位置までのオフセット
　読み込み位置が無効の場合：0(showmanyc() を呼び出します)

int_type streambuf::snextc()

一文字読み込みます。読み込んだ文字が eof でなければ、次の一文字を読み込みます。
リターン値は次のとおりです。
　eof でない場合：読み込んだ文字
　eof の場合：eof

int_type streambuf::sbumpc()

一文字読み込みポインタを次に設定します。
リターン値は次のとおりです。
　読み込み位置が無効でない場合：読み込んだ文字
　読み込み位置が無効の場合：eof

int_type streambuf::sgetc()

一文字読み込みます。
リターン値は次のとおりです。
　読み込み位置が無効でない場合：読み込んだ文字
　読み込み位置が無効の場合：eof

int streambuf::sgetn(char* s, streamsize n)

s の指す記憶領域に n 個の文字を設定します。
文字列中に eof を検出した場合、設定を終了します。
リターン値は、設定した文字数です。

int_type streambuf::sputbackc(char c)

読み込み位置が正常で読み込み位置のプットバックデータが c と同一の場合、読み込み位置をプットバックします。
リターン値は次のとおりです。
　プットバックできた場合：c の値
　プットバックできなかった場合：eof

int streambuf::sungetc()

読み込み位置が正常である場合、読み込み位置をプットバックします。
リターン値は次のとおりです。
　プットバックできた場合：プットバックした値
　プットバックできなかった場合：eof

int streambuf::sputc(char c)

文字 c を挿入します。
リターン値は次のとおりです。
　書き込み位置が正しい場合：c の値
　書き込み位置が不正な場合：eof

int_type streambuf::sputn(const char* s, streamsize n)

s の指す n 個の文字を挿入します。
バッファが n より小さい場合は、バッファサイズ分だけ挿入します。
リターン値は、挿入された文字数です。

char* streambuf::eback() const

入力ストリームの先頭ポインタを求めます。
リターン値は、先頭ポインタです。

char* streambuf::gptr() const

入力ストリームの次ポインタを求めます。
リターン値は、次ポインタです。

char* streambuf::egptr() const

入力ストリームの最後尾ポインタを求めます。
リターン値は、最後尾ポインタです。

void streambuf::gbump(int n)

入力ストリームの次ポインタを n 進めます。

void streambuf::setg(char* gbeg, char* gnext, char* gend)

入力ストリームの各ポインタに、以下の設定を行います。
*B_beg_pptr=gbeg;
*B_next_pptr=gnext;
B_end_ptr=gend;
*_B_cnt_ptr=gend-gnext;
*_B_len_ptr=gend-gbeg;

char* streambuf::pbase() const

出力ストリームの先頭ポインタを求めます。
リターン値は、先頭ポインタです。

char* streambuf::pptr() const

出力ストリームの次ポインタを求めます。
リターン値は、次ポインタです。

char* streambuf::epptr() const

出力ストリームの最後尾ポインタを求めます。
リターン値は、最後尾ポインタです。

void streambuf::pbump(int n)

出力ストリームの次ポインタを n 進めます。

```
void streambuf::setp(char* pbeg, char* pend)
```

出力ストリームの各ポインタに、以下の設定を行います。
*B_beg_pptr=pbeg;
*B_next_pptr=pbeg;
B_end_ptr=pend;
*_B_cnt_ptr=pend-pbeg;
*_B_len_ptr=pend-pbeg;

```
virtual streambuf* streambuf::setbuf(char* s, streamsize n)
```

streambuf から派生する各クラスごとに、個別に定義する演算を実行します。
リターン値は *this です。このクラスでは処理を定義していません。

```
virtual pos_type streambuf::seekoff(off_type off, ios_base::seekdir way, ios_base::openmode =
(ios_base::openmode)(ios_base::in | ios_base::out))
```

ストリーム位置を変更します。
リターン値は -1 です。このクラスでは処理を定義していません。

```
virtual pos_type streambuf::seekpos(pos_type sp, ios_base::openmode =
(ios_base::openmode)(ios_base::in | ios_base::out))
```

ストリーム位置を変更します。
リターン値は -1 です。このクラスでは処理を定義していません。

```
virtual int streambuf::sync()
```

出力ストリームをフラッシュします。
リターン値は 0 です。このクラスでは処理を定義していません。

```
virtual int streambuf::showmanyc()
```

入力ストリームの有効な文字数を求めます。
リターン値は 0 です。このクラスでは処理を定義していません。

```
virtual streamsize streambuf::xsgetn(char* s, streamsize n)
```

s の指す記憶領域に n 個の文字を設定します。
バッファが n より小さい場合は、バッファサイズ分だけ設定します。
リターン値は、入力された文字数です。

```
virtual int_type streambuf::underflow()
```

ストリーム位置を動かさずに一文字読み込みます。
リターン値は eof です。このクラスでは処理を定義していません。

```
virtual int_type streambuf::uflow()
```

次ポインタの一文字を読み込みます。
リターン値は eof です。このクラスでは処理を定義していません。

virtual int_type streambuf::pbackfail(int_type c = eof)

c によって示される文字をプットバックします。
リターン値は eof です。このクラスでは処理を定義していません。

virtual streamsize streambuf::xsputn(const char* s, streamsize n)

s の指す n 個の文字を挿入します。
バッファが n より小さい場合は、バッファサイズ分だけ挿入します。
リターン値は、挿入された文字数です。

virtual int_type streambuf::overflow(int_type c = eof)

c を出力ストリームに挿入します。
リターン値は eof です。このクラスでは処理を定義していません。

(f) istream::sentry クラス

種別	定義名	説明
変数	ok_	入力可能状態か否かを意味します。
関数	sentry(istream& is, bool noskipws = false)	コンストラクタです。
	~sentry()	デストラクタです。
	operator bool()	ok_ を参照します。

istream::sentry::sentry(istream& is, bool noskipws = _false)

内部クラス sentry のコンストラクタです。
good() が非 0 の場合、フォーマット付きまたはフォーマットなし入力を可能にします。
tie() が非 0 の場合、関連する出力ストリームをフラッシュします。

istream::sentry::~sentry()

内部クラス sentry のデストラクタです。

istream::sentry::operator bool()

ok_ を参照します。
リターン値は ok_ です。

(g) istream クラス

種別	定義名	説明
変数	chcount	最後にコールされた入力関数が抽出した文字数です。
関数	int _ec2p_getistr(char* str, unsigned int dig, int mode)	str を dig が示す基数で変換します。
	istream(streambuf* sb)	コンストラクタです。
	virtual ~istream()	デストラクタです。
	istream& operator>>(bool& n)	抽出した文字を n に格納します。
	istream& operator>>(short& n)	
	istream& operator>>(unsigned short& n)	
	istream& operator>>(int& n)	
	istream& operator>>(unsigned int& n)	
	istream& operator>>(long& n)	
	istream& operator>>(unsigned long& n)	
	istream& operator>>(long long& n)	
	istream& operator>>(unsigned long long& n)	
	istream& operator>>(float& n)	
	istream& operator>>(double& n)	
	istream& operator>>(long double& n)	
	istream& operator>>(void*& p)	void を指すポインタに変換して p に格納します。
	istream& operator>>(streambuf* sb)	文字を抽出し、sb の指す記憶領域へ格納します。
	streamsize gcount() const	chcount(抽出文字数) を求めます。
	int_type get()	文字を抽出します。
	istream& get(char& c)	文字を抽出し c に格納します。
	istream& get(signed char& c)	
	istream& get(unsigned char& c)	

種別	定義名	説明
関数	istream& get(char* s, streamsize n)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
	istream& get(signed char* s, streamsize n)	
	istream& get(unsigned char* s, streamsize n)	
	istream& get(char* s, streamsize n, char delim)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。文字列内に 'delim' を検出したら、入力を終了します。
	istream& get(signed char* s, streamsize n, char delim)	
	istream& get(unsigned char* s, streamsize n, char delim)	
	istream& get(streambuf& sb)	文字列を抽出し、sb の指す記憶領域に格納します。
	istream& get(streambuf& sb, char delim)	文字列を抽出し、sb の指す記憶領域に格納します。途中で文字 'delim' を検出したら、入力を終了します。
	istream& getline(char* s, streamsize n)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
	istream& getline(signed char* s, streamsize n)	
	istream& getline(unsigned char* s, streamsize n)	
	istream& getline(char* s, streamsize n, char delim)	サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。途中で文字 'delim' を検出したら、入力を終了します。
	istream& getline(signed char* s, streamsize n, char delim)	
	istream& getline(unsigned char* s, streamsize n, char delim)	
	istream& ignore(streamsize n = 1, int_type delim = streambuf::eof)	n 個の文字を読み飛ばします。途中で文字 'delim' を検出したら、読み飛ばし処理を中止します。
	int_type peek()	次の入手可能な入力文字を求めます。
	istream& read(char* s, streamsize n)	サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
	istream& read(signed char* s, streamsize n)	
	istream& read(unsigned char* s, streamsize n)	
	streamsize readsome(char* s, streamsize n)	サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
	streamsize readsome(signed char* s, streamsize n)	
	streamsize readsome(unsigned char* s, streamsize n)	
	istream& putback(char c)	文字を入力ストリームに戻します。
	istream& unget()	入力ストリームの位置を戻します。

種別	定義名	説明
関数	int sync()	入力ストリームがあるかどうかを調べます。この関数は streambuf::pubsync() を呼び出します。
	pos_type tellg()	入力ストリームの位置を調べます。この関数は streambuf::pubseekoff(0,cur,in) を呼び出します。
	istream& seekg(pos_type pos)	現在のストリームポインタから pos だけ移動します。この関数は streambuf::pubseekpos(pos) を呼び出します。
	istream& seekg(off_type off, ios_base::seekdir dir)	dir で指定された方法で入力ストリームの読み込み位置を移動します。この関数は streambuf::pubseekoff(off,dir) を呼び出します。

```
int istream::_ec2p_getistr(char* str, unsigned int dig, int mode)
```

str を dig が示す基数で変換します。
リターン値は、変換した基数です。

```
istream::istream(streambuf* sb)
```

クラス istream のコンストラクタです。
ios::init(sb) を呼び出します。
chcount=0 の設定を行います。

```
virtual istream::~istream()
```

クラス istream のデストラクタです。

```
istream& istream::operator>>(bool& n)
```

抽出した文字を n に格納します。
リターン値は *this です。

```
istream& istream::operator>>(short& n)
```

抽出した文字を n に格納します。
リターン値は *this です。

```
istream& istream::operator>>(unsigned short& n)
```

抽出した文字を n に格納します。
リターン値は *this です。

```
istream& istream::operator>>(int& n)
```

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(unsigned int& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(long& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(unsigned long& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(long long& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(unsigned long long& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(float& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(double& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(long double& n)

抽出した文字を n に格納します。
リターン値は *this です。

istream& istream::operator>>(void*& p)

抽出した文字を void* 型に変換し、p の指す記憶領域に格納します。
リターン値は *this です。

istream& istream::operator>>(streambuf* sb)

文字を抽出し、sb の指す記憶領域に格納します。
抽出文字がない場合は、setstate(failbit) を呼び出します。
リターン値は *this です。

streamsize istream::gcount() const

chcount(抽出文字数) を参照します。
リターン値は chcount です。

int_type istream::get()

文字を抽出します。
リターン値は次のとおりです。
抽出可能の場合：抽出した文字
抽出不可の場合：setstate(failbat) を呼び出して、streambuf::eof

istream& istream::get(char& c)

文字を抽出し c に格納します。抽出した文字が streambuf::eof の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(signed char& c)

文字を抽出し c に格納します。抽出した文字が streambuf::eof の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(unsigned char& c)

文字を抽出し c に格納します。抽出した文字が streambuf::eof の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(signed char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(unsigned char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
文字列内に 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(signed char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
文字列内に 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(unsigned char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
文字列内に 'delim' を検出したら、終了します。
ok_==false または抽出した文
字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(streambuf& sb)

文字列を抽出し、sb の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::get(streambuf& sb, char delim)

文字列を抽出し、sb の指す記憶領域に格納します。
途中で文字 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(signed char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(unsigned char* s, streamsize n)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
途中で文字 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(signed char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
途中で文字 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::getline(unsigned char* s, streamsize n, char delim)

サイズ n-1 の文字列を抽出し、s の指す記憶領域に格納します。
途中で文字 'delim' を検出したら、終了します。
ok_==false または抽出した文字数が 0 の場合は、failbit を設定します。
リターン値は *this です。

istream& istream::ignore(streamsize n = 1, int_type delim = streambuf::eof)

n 個の文字を読み飛ばします。
途中で文字 'delim' を検出したら、読み飛ばし処理を中止します。
リターン値は *this です。

int_type istream::peek()

次の入力可能な入力文字を求めます。
リターン値は次のとおりです。
ok_==false の場合：streambuf::eof
ok_!=false の場合：rdbuf()->sgetc()

istream& istream::read(char* s, streamsize n)

ok_!=false の場合、サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
抽出した文字数が n と異なる場合、eofbit を設定します。
リターン値は *this です。

istream& istream::read(signed char* s, streamsize n)

ok_!=false の場合、サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
抽出した文字数が n と異なる場合、eofbit を設定します。
リターン値は *this です。

istream& istream::read(unsigned char* s, streamsize n)

ok_!=false の場合、サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
抽出した文字数が n と異なる場合、eofbit を設定します。
リターン値は *this です。

streamsize istream::readsome(char* s, streamsize n)

サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
文字数がストリームサイズより大きければ、ストリームサイズ分格納します。
リターン値は、抽出した文字数です。

streamsize istream::readsome(signed char* s, streamsize n)

サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
文字数がストリームサイズより大きければ、ストリームサイズ分格納します。
リターン値は、抽出した文字数です。

streamsize istream::readsome(unsigned char* s, streamsize n)

サイズ n の文字列を抽出し、s の指す記憶領域に格納します。
文字数がストリームサイズより大きければ、ストリームサイズ分格納します。
リターン値は、抽出した文字数です。

istream& istream::putback(char c)

文字 c を入力ストリームに戻します。プットバックした文字が streambuf::eof の場合は、badbit を設定します。
リターン値は *this です。

istream& istream::unget()

入力ストリームのポインタをひとつ戻します。
抽出した文字が streambuf::eof の場合、badbit を設定します。
リターン値は *this です。

int istream::sync()

入力ストリームがあるかどうかを調べます。
この関数は streambuf::pubsync() を呼び出します。
リターン値は次のとおりです。
　入力ストリームがない場合：streambuf::eof
　入力ストリームがある場合：0

pos_type istream::tellg()

入力ストリームの位置を調べます。
この関数は streambuf::pubseekoff(0,cur,in) を呼び出します。
リターン値は次のとおりです。
　ストリームの先頭からのオフセット
　ただし、入力処理にエラーが発生した場合は -1

istream& istream::seekg(pos_type pos)

現在のストリームポインタから pos だけ移動します。
この関数は streambuf::pubseekpos(pos) を呼び出します。
リターン値は *this です。

istream& istream::seekg(off_type off, ios_base::seekdir dir)

dir で指定された方法で入力ストリームの読み込み位置を移動します。
この関数は streambuf::pubseekoff(off,dir) を呼び出します。
入力処理にエラーがある場合は処理は行いません。
リターン値は *this です。

(h) istream クラスマニピュレータ

種別	定義名	説明
関数	istream& ws(istream& is)	空白類を読み飛ばします。

istream& ws(istream& is)

空白類を読み飛ばします。
リターン値は is です。

(i) istream メンバ外関数

種別	定義名	説明
関数	istream& operator>>(istream& in, char* s)	文字列を抽出し、s の指す記憶領域に格納します。
	istream& operator>>(istream& in, signed char* s)	
	istream& operator>>(istream& in, unsigned char* s)	
	istream& operator>>(istream& in, char& c)	文字を抽出し、c に格納します。
	istream& operator>>(istream& in, singed char& c)	
	istream& operator>>(istream& in, unsigned char& c)	

istream& operator>>(istream& in, char* s)

文字列を抽出し、s の指す記憶領域に格納します。
(フィールド幅 -1) 個の文字を格納したか、または入力ストリームに streambuf::eof が現れたか、または次の入力可能な文字 c が isspace(c)==1 の場合、処理は終了します。格納文字数が 0 の場合は failbit を設定します。
リターン値は in です。

istream& operator>>(istream& in, signed char* s)

文字列を抽出し、s の指す記憶領域に格納します。
(フィールド幅 -1) 個の文字を格納したか、または入力ストリームに streambuf::eof が現れたか、または次の入力可能な文字 c が isspace(c)==1 の場合、処理は終了します。格納文字数が 0 の場合は failbit を設定します。
リターン値は in です。

istream& operator>>(istream& in, unsigned char* s)

文字列を抽出し、s の指す記憶領域に格納します。
(フィールド幅 -1) 個の文字を格納したか、または入力ストリームに streambuf::eof が現れたか、または次の入力可能な文字 c が isspace(c)==1 の場合、処理は終了します。格納文字数が 0 の場合は failbit を設定します。
リターン値は in です。

istream& operator>>(istream& in, char& c)

文字を抽出し、c に格納します。
抽出入力がない場合、failbit を設定します。
リターン値は in です。

istream& operator>>(istream& in, singed char& c)

文字を抽出し、c に格納します。
抽出入力がない場合、failbit を設定します。
リターン値は in です。

istream& operator>>(istream& in, unsigned char& c)

文字を抽出し、c に格納します。
抽出入力がない場合、failbit を設定します。
リターン値は in です。

(j) ostream::sentry クラス

種別	定義名	説明
変数	ok_	出力可能状態か否かを意味します。
	_ _ec2p_os	ostream オブジェクトへのポインタです。
関数	sentry(ostream& os)	コンストラクタです。
	~sentry()	デストラクタです。
	operator bool()	ok_ を参照します。

ostream::sentry::sentry(ostream& os)

内部クラス sentry のコンストラクタです。
good() が非 0 かつ tie() が非 0 なら flush() を呼び出します。_ _ec2p_os に os を設定します。

ostream::sentry::~sentry()

内部クラス sentry のデストラクタです。
_ _ec2p_os->flags() & ios_base::unitbuf が真なら、flush() を呼び出します。

ostream::sentry::operator bool()

ok_ を参照します。
リターン値は ok_ です。

(k) ostream クラス

種別	定義名	説明
関数	ostream(streambuf* sbptr)	コンストラクタです。
	virtual ~ostream()	デストラクタです。
	ostream& operator<<(bool n)	n を出力ストリームに挿入します。
	ostream& operator<<(short n)	
	ostream& operator<<(unsigned short n)	
	ostream& operator<<(int n)	
	ostream& operator<<(unsigned int n)	
	ostream& operator<<(long n)	
	ostream& operator<<(unsigned long n)	
	ostream& operator<<(long long n)	
	ostream& operator<<(unsigned long long n)	
	ostream& operator<<(float n)	
	ostream& operator<<(double n)	
	ostream& operator<<(long double n)	
	ostream& operator<<(void* n)	
	ostream& operator<<(streambuf* sbptr)	sbptr の出力列を出力ストリームに挿入します。
	ostream& put(char c)	文字 c を出力ストリームに挿入します。
	ostream& write(const char* s, streamsize n)	s の n 個の文字を出力ストリームに挿入します。
	ostream& write(const signed char* s, streamsize n)	
	ostream& write(const unsigned char* s, streamsize n)	
	ostream& flush()	出力ストリームをフラッシュします。この関数は streambuf::pubsync() を呼び出します。
	pos_type tellp()	現在の書き込み位置を求めます。この関数は streambuf::pubseekoff(0,cur,out) を呼び出します。
	ostream& seekp(pos_type pos)	ストリームの先頭から現在の位置までのオフセットを求めます。現在のストリームポインタから pos だけ移動します。この関数は streambuf::pubseekpos(pos) を呼び出します。
	ostream& seekp(off_type off, seekdir dir)	dir を基準として、ストリームの書き込み位置を off 分だけ移動します。この関数は streambuf::pubseekoff(off,dir) を呼び出します。

ostream::ostream(streambuf* sbptr)

コンストラクタです。
ios(sbptr) を呼び出します。

virtual ostream::~ostream()

デストラクタです。

ostream& ostream::operator<<(bool n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(short n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(unsigned short n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(int n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(unsigned int n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(long n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(unsigned long n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(long long n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(unsigned long long n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(float n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(double n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(long double n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(void* n)

sentry::ok_==true のとき、n を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::operator<<(streambuf* sbptr)

sentry::ok_==true のとき、sbptr の出力列を出力ストリームに挿入します。
sentry::ok_==false のとき、failbit を設定します。
リターン値は *this です。

ostream& ostream::put(char c)

sentry::ok_==true かつ rdbuf()->sputc(c)!=streambuf::eof のとき、c を出力ストリームに挿入します。
上記以外のとき、badbit を設定します。
リターン値は *this です。

ostream& ostream::write(const char* s, streamsize n)

sentry::ok_==true かつ rdbuf()->sputn(s, n)==n のとき、s の n 個の文字を出力ストリームに挿入します。
上記以外のとき、badbit を設定します。
リターン値は *this です。

ostream& ostream::write(const signed char* s, streamsize n)

sentry::ok_==true かつ rdbuf()->sputn(s, n)==n のとき、s の n 個の文字を出力ストリームに挿入します。
上記以外のとき、badbit を設定します。
リターン値は *this です。

ostream& ostream::write(const unsigned char* s, streamsize n)

sentry::ok_==true かつ rdbuf()->sputn(s, n)==n のとき、s の n 個の文字を出力ストリームに挿入します。
上記以外のとき、badbit を設定します。
リターン値は *this です。

ostream& ostream::flush()

出力ストリームをフラッシュします。
この関数は streambuf::pubsync() を呼び出します。
リターン値は *this です。

pos_type ostream::tellp()

現在の書き込み位置を求めます。
この関数は streambuf::pubseekoff(0,cur,out) を呼び出します。
リターン値は次のとおりです。
　現在のストリームの位置
　ただし、処理中にエラーが発生した場合は -1

ostream& ostream::seekp(pos_type pos)

エラーがないとき、ストリームの先頭から現在の位置までのオフセットを求めます。
また、現在のストリームポインタから pos だけ移動します。
この関数は streambuf::pubseekpos(pos) を呼び出します。
リターン値は *this です。

ostream& ostream::seekp(off_type off, seekdir dir)

エラーがないとき、dir を基準として off 分ストリームの位置を移動します。
この関数は streambuf::pubseekoff(off,dir) を呼び出します。
リターン値は *this です。

(l) ostream クラスマニピュレータ

種別	定義名	説明
関数	ostream& endl(ostream& os)	改行を挿入し、出力ストリームをフラッシュします。
	ostream& ends(ostream& os)	ヌルコードを挿入します。
	ostream& flush(ostream& os)	出力ストリームをフラッシュします。

ostream& endl(ostream& os)

ストリームに改行文字を挿入します。
出力ストリームをフラッシュします。この関数は flush() を呼び出します。
リターン値は os です。

ostream& ends(ostream& os)

出力ストリームにヌルコードを挿入します。
リターン値は os です。

ostream& flush(ostream& os)

出力ストリームをフラッシュします。この関数は streambuf::sync() を呼び出します。
リターン値は os です。

(m) ostream メンバ外関数

種別	定義名	説明
関数	ostream& operator<<(ostream& os, char s)	s を出力ストリームに挿入します。
	ostream& operator<<(ostream& os, signed char s)	
	ostream& operator<<(ostream& os, unsigned char s)	
	ostream& operator<<(ostream& os, const char* s)	
	ostream& operator<<(ostream& os, const singed char* s)	
	ostream& operator<<(ostream& os, const unsigned char* s)	

ostream& operator<<(ostream& os, char s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

ostream& operator<<(ostream& os, signed char s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

ostream& operator<<(ostream& os, unsigned char s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

ostream& operator<<(ostream& os, const char* s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

ostream& operator<<(ostream& os, const singed char* s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

ostream& operator<<(ostream& os, const unsigned char* s)

sentry::ok_==true かつエラーがないとき、s を出力ストリームに挿入します。
上記以外のとき、failbit を設定します。
リターン値は os です。

(n) smanip クラスマニピュレータ

種別	定義名	説明
関数	smanip resetiosflags(ios_base::fmtflags mask)	mask 値で指定されたフラグをクリアします。
	smanip setiosflags(ios_base::fmtflags mask)	書式フラグ (fmtfl) を設定します。
	smanip setbase(int base)	出力時に用いる基数を設定します。
	smanip setfill(char c)	詰め文字 (fillch) を設定します。
	smanip setprecision(int n)	精度 (prec) を設定します。
	smanip setw(int n)	フィールド幅 (wide) を設定します。

smanip resetiosflags(ios_base::fmtflags mask)

mask 値で指定されたフラグをクリアします。
リターン値は、入出力対象のオブジェクトです。

smanip setiosflags(ios_base::fmtflags mask)

書式フラグ (fmtfl) を設定します。
リターン値は、入出力対象のオブジェクトです。

smanip setbase(int base)

出力時に用いる基数を設定します。
リターン値は、入出力対象のオブジェクトです。

smanip setfill(char c)

詰め文字 (fillch) を設定します。
リターン値は、入出力対象のオブジェクトです。

smanip setprecision(int n)

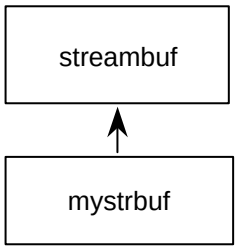
精度 (prec) を設定します。
リターン値は、入出力対象のオブジェクトです。

smanip setw(int n)

フィールド幅 (wide) を設定します。
リターン値は、入出力対象のオブジェクトです。

(o) EC++ 入出力ライブラリの使用例
istream, ostream のオブジェクトの初期化時に streambuf のかわりに mystrbuf クラスのオブジェクトへのポインタを使うことにより入出力ストリームが使用可能になります。

クラスの派生関係は次のようになります。矢印は、派生クラスから基底クラスを参照していることを示します。



種別	定義名	説明
変数	_file_Ptr	ファイルポインタです。
関数	mystrbuf)	コンストラクタです。streambuf バッファの初期化を行います。
	mystrbuvoid* ptr)	
	virtual ~mystrbuf()	デストラクタです。
	void* myfptr) const	FILE 型構造体へのポインタを返します。
	mystrbuf* openconst char* filename, int mode)	ファイル名とモードを指定して、ファイルをオープンします。
	mystrbuf* close()	ファイルのクローズを行います。
	virtual streambuf* setbuf(char* s, streamsize n)	ストリーム入出力用のバッファを確保します。
	virtual pos_type seekoff(off_type off, ios_base::seekdir way, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	ストリームポインタの位置を変えます。
	virtual pos_type seekpos(pos_type sp, ios_base::openmode = (ios_base::openmode) (ios_base::in | ios_base::out))	ストリームポインタの位置を変えます。
	virtual int sync()	ストリームをフラッシュします。
	virtual int showmanyc()	入力ストリームの有効な文字数を返します。
	virtual int_type underflow()	ストリーム位置を動かさずに一文字読み込みます。
	virtual int_type pbackfail(int_type c = streambuf::eof)	c によって示される文字をプットバックします。
	virtual int_type overflow(int_type c = streambuf::eof)	c によって示される文字を挿入します。
	void _Init(_f_type* fp)	初期処理です。

例

```
#include <istream>
#include <ostream>
#include <mystrbuf>
#include <string>
#include <new>
#include <stdio.h>
void main(void)
{
    mystrbuf myfin(stdin);
    mystrbuf myfout(stdout);
    istream mycin(&myfin);
    ostream mycout(&myfout);

    int i;
    short s;
    long l;
    char c;
    string str;

    mycin >> i >> s >> l >> c >> str;
    mycout << "This is EC++ Library." << endl
           << i << s << l << c << str << endl;
    return;
}
```

7.5.2 メモリ管理用ライブラリ

メモリの管理用ライブラリに対応するヘッダファイルは以下の通りです。

- <new>
  メモリの確保・解放を行う関数を定義します。

_ec2p_new_handler 変数に例外処理関数のアドレスを設定することにより、メモリ確保に失敗した場合、例外処理を実行することができます。
_ec2p_new_handler は static 変数で、初期値は NULL です。このハンドラを使用することにより、リエントラント性は失われます。

例外処理関数に要求される動作：

- 割り当て可能な領域を作成して返します。
- 作成できない場合の動作は規定されていません。

種別	定義名	説明
型	new_handler	void 型を返す関数へのポインタ型です。
変数	_ec2p_new_handler	例外処理関数へのポインタです。

種別	定義名	説明
関数	void* operator new(size_t size)	size 分の領域を確保します。
	void* operator new[ ](size_t size)	size 分の配列領域を確保します。
	void* operator new(size_t size, void* ptr)	ptr の指している領域を記憶領域として割り当てます。
	void* operator new[ ](size_t size, void* ptr)	ptr の指している領域を配列領域として割り当てます。
	void operator delete(void* ptr)	領域を解放します。
	void operator delete[ ](void* ptr)	配列領域を解放します。
	new_handler set_new_handler(new_handler new_P)	_ec2p_new_handler に例外処理関数アドレス (new_P) を設定します。

void* operator new(size_t size)

size バイト分の領域を割り当てます。
領域割り当てに失敗し、かつ new_handler が設定されていれば、new_handler を呼び出します。
リターン値は次のとおりです。
　領域確保に成功した場合：void 型へのポインタ
　領域確保に失敗した場合：NULL

void* operator new[ ](size_t size)

size 分の配列領域を確保します。
領域割り当てに失敗し、かつ new_handler が設定されていれば、new_handler を呼び出します。
リターン値は次のとおりです。
　領域確保に成功した場合：void 型へのポインタ
　領域確保に失敗した場合：NULL

void* operator new(size_t size, void* ptr)

ptr の指している領域を記憶領域として割り当てます。
リターン値は ptr です。

void* operator new[ ](size_t size, void* ptr)

ptr の指している領域を配列領域として割り当てます。
リターン値は ptr です。

void operator delete(void* ptr)

ptr が指す記憶領域を解放します。ptr が NULL のときは何もしません。

void operator delete[ ](void* ptr)

ptr が指す配列領域を解放します。ptr が NULL のときは何もしません。

new_handler set_new_handler(new_handler new_P)

_ec2p_new_handler に new_P を設定します。
リターン値は _ec2p_new_handler です。

7.5.3 複素数計算用クラスライブラリ

複素数計算用クラスライブラリに対応するヘッダファイルは以下のとおりです。

- <complex>
  float_complex クラス、double_complex クラスを定義します。

これらのクラスには派生関係はありません。

(a) float_complex クラス

種別	定義名	説明
型	value_type	float 型です。
変数	_re	float 精度の実数部を定義します。
	_im	float 精度の虚数部を定義します。
関数	float_complex(float re = 0.0f, float im = 0.0f)	コンストラクタです。
	float_complex(const double_complex& rhs)	
	float real() const	実数部 (_re) を求めます。
	float imag() const	虚数部 (_im) を求めます。
	float_complex& operator=(float rhs)	rhs を実数部にコピーします。虚数部は 0.0f を設定します。
	float_complex& operator+=(float rhs)	rhs を実数部に加算し、和を *this に格納します。
	float_complex& operator-=(float rhs)	rhs を実数部から減算し、差を *this に格納します。
	float_complex& operator*=(float rhs)	rhs を乗算し、積を *this に格納します。
	float_complex& operator/=(float rhs)	rhs で除算し、商を *this に格納します。
	float_complex& operator=(const float_complex& rhs)	rhs をコピーします。
	float_complex& operator+=(const float_complex& rhs)	rhs を加算し、和を *this に格納します。
	float_complex& operator-=(const float_complex& rhs)	rhs を減算し、差を *this に格納します。
	float_complex& operator*=(const float_complex& rhs)	rhs を乗算し、積を *this に格納します。
	float_complex& operator/=(const float_complex& rhs)	rhs で除算し、商を *this に格納します。

float_complex::float_complex(float re = 0.0f, float im = 0.0f)

クラス float_complex のコンストラクタです。
以下の値で初期化します。
_re = re;
_im = im;

float_complex::float_complex(const double_complex& rhs)

クラス float_complex のコンストラクタです。
以下の値で初期化します。
_re = (float)rhs.real();
_im = (float)rhs.imag();

float float_complex::real() const

実数部を求めます。
リターン値は、this->_re です。

float float_complex::imag() const

虚数部を求めます。
リターン値は、this->_im です。

float_complex& float_complex::operator=(float rhs)

rhs を実数部 (_re) にコピーします。虚数部 (_im) は 0.0f を設定します。
リターン値は *this です。

float_complex& float_complex::operator+=(float rhs)

rhs を実数部 (_re) に加算し、結果を実数部 (_re) に格納します。虚数部 (_im) の値は変わりません。
リターン値は *this です。

float_complex& float_complex::operator-=(float rhs)

rhs を実数部 (_re) から減算し、結果を実数部 (_re) に格納します。虚数部 (_im) の値は変わりません。
リターン値は *this です。

float_complex& float_complex::operator*=(float rhs)

rhs と乗算し、結果を *this に格納します。
(_re=_re*rhs, _im=_im*rhs)
リターン値は *this です。

float_complex& float_complex::operator/=(float rhs)

rhs で除算し、結果を *this に格納します。
(_re=_re/rhs, _im=_im/rhs)
リターン値は *this です。

float_complex& float_complex::operator=(const float_complex& rhs)

rhs をコピーします。
リターン値は *this です。

float_complex& float_complex::operator+=(const float_complex& rhs)

rhs を加算し、結果を *this に格納します。
リターン値は *this です。

float_complex& float_complex::operator-=(const float_complex& rhs)

rhs を減算し、結果を *this に格納します。
リターン値は *this です。

float_complex& float_complex::operator*=(const float_complex& rhs)

rhs と乗算し、結果を *this に格納します。
リターン値は *this です。

float_complex& float_complex::operator/=(const float_complex& rhs)

rhs で除算し、結果を *this に格納します。
リターン値は *this です。

(b) float_complex メンバ外関数

種別	定義名	説明
関数	float_complex operator+(const float_complex& lhs)	lhs の単項＋演算を行います。
	float_complex operator+(const float_complex& lhs, const float_complex& rhs)	lhs と rhs を加算した結果を返却します。
	float_complex operator+(const float_complex& lhs, const float& rhs)	
	float_complex operator+(const float& lhs, const float_complex& rhs)	
	float_complex operator-(const float_complex& lhs)	lhs の単項－演算を行います。
	float_complex operator-(const float_complex& lhs, const float_complex& rhs)	lhs から rhs を減算した結果を返却します。
	float_complex operator-(const float_complex& lhs, const float& rhs)	
	float_complex operator-(const float& lhs, const float_complex& rhs)	
	float_complex operator*(const float_complex& lhs, const float_complex& rhs)	lhs と rhs を乗算した結果を返却します。
	float_complex operator*(const float_complex& lhs, const float& rhs)	
	float_complex operator*(const float& lhs, const float_complex& rhs)	
	float_complex operator/(const float_complex& lhs, const float_complex& rhs)	lhs を rhs で除算した結果を返却します。
	float_complex operator/(const float_complex& lhs, const float& rhs)	
	float_complex operator/(const float& lhs, const float_complex& rhs)	

種別	定義名	説明
関数	bool operator==(const float_complex& lhs, const float_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator==(const float_complex& lhs, const float& rhs)	
	bool operator==(const float& lhs, const float_complex& rhs)	
	bool operator!=(const float_complex& lhs, const float_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator!=(const float_complex& lhs, const float& rhs)	
	bool operator!=(const float& lhs, const float_complex& rhs)	
	istream& operator>>(istream& is, float_complex& x)	u,(u),または(u,v) (u:実数部、v:虚数部)形式のxを入力します。
	ostream& operator<<(ostream& os, float_complex& x)	xをu,(u)または (u,v) (u:実数部、v:虚数部)形式で出力します。
	float real(const float_complex& x)	実数部を求めます。
	float imag(const float_complex& x)	虚数部を求めます。
	float abs(const float_comlex& x)	絶対値を求めます。
	float arg(const float_complex& x)	位相角度を求めます。
	float norm(const float_complex& x)	2 乗の絶対値を求めます。
	float_complex conj(const float_complex& x)	共役複素数を求めます。
	float_complex polar(const float& rho, const float& theta)	大きさが rho で位相角度が theta の複素数に対応する float_complex 値を求めます。
	float_complex cos(const float_complex& x)	複素余弦を求めます。
	float_complex cosh(const float_complex& x)	複素双曲余弦を求めます。
	float_complex exp(const float_complex& x)	指数関数を求めます。
	float_complex log(const float_complex& x)	自然対数を求めます。
	float_complex log10(const float_complex& x)	常用対数を求めます。

種別	定義名	説明
関数	float_complex pow(const float_complex& x, int y)	x の y 乗を求めます。
	float_complex pow(const float_complex& x, const float& y)	
	float_complex pow(const float_complex& x, const float_complex& y)	
	float_complex pow(const float& x, const float_complex& y)	
	float_complex sin(const float_complex& x)	複素正弦を求めます。
	float_complex sinh(const float_complex& x)	複素双曲正弦を求めます。
	float_complex sqrt(const float_complex& x)	右半空間における範囲での平方根を求めます。
	float_complex tan(const float_complex& x)	複素正接を求めます。
	float_complex tanh(const float_complex& x)	複素双曲正接を求めます。

float_complex operator+(const float_complex& lhs)

lhs の単項＋演算を行います。
リターン値は lhs です。

float_complex operator+(const float_complex& lhs, const float_complex& rhs)

lhs と rhs を加算した結果を返却します。
リターン値は、float_complex(lhs)+=rhs です。

float_complex operator+(const float_complex& lhs, const float& rhs)

lhs と rhs を加算した結果を返却します。
リターン値は、float_complex(lhs)+=rhs です。

float_complex operator+(const float& lhs, const float_complex& rhs)

lhs と rhs を加算した結果を返却します。
リターン値は、float_complex(lhs)+=rhs です。

float_complex operator-(const float_complex& lhs)

lhs の単項－演算を行います。
リターン値は、float_complex(-lhs.real(),-lhs.imag()) です。

float_complex operator-(const float_complex& lhs, const float_complex& rhs)

lhs から rhs を減算した結果を返却します。
リターン値は、float_complex(lhs)-=rhs です。

float_complex operator-(const float_complex& lhs, const float& rhs)

lhs から rhs を減算した結果を返却します。
リターン値は、float_complex(lhs)-=rhs です。

float_complex operator-(const float& lhs, const float_complex& rhs)

lhs から rhs を減算した結果を返却します。
リターン値は、float_complex(lhs)-=rhs です。

float_complex operator*(const float_complex& lhs, const float_complex& rhs)

lhs と rhs を乗算した結果を返却します。
リターン値は、float_complex(lhs)*=rhs です。

float_complex operator*(const float_complex& lhs, const float& rhs)

lhs と rhs を乗算した結果を返却します。
リターン値は、float_complex(lhs)*=rhs です。

float_complex operator*(const float& lhs, const float_complex& rhs)

lhs と rhs を乗算した結果を返却します。
リターン値は、float_complex(lhs)*=rhs です。

float_complex operator/(const float_complex& lhs, const float_complex& rhs)

lhs を rhs で除算した結果を返却します。
リターン値は、float_complex(lhs)/=rhs です。

float_complex operator/(const float_complex& lhs, const float& rhs)

lhs を rhs で除算した結果を返却します。
リターン値は、float_complex(lhs)/=rhs です。

float_complex operator/(const float& lhs, const float_complex& rhs)

lhs を rhs で除算した結果を返却します。
リターン値は、float_complex(lhs)/=rhs です。

bool operator==(const float_complex& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator==(const float_complex& lhs, const float& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator==(const float& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator!=(const float_complex& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

bool operator!=(const float_complex& lhs, const float& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

bool operator!=(const float& lhs, const float_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。float 型引数の場合、虚数部は float 型の 0.0f と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

istream& operator>>(istream& is, float_complex& x)

u,(u), または (u,v) (u は実数部、v は虚数部) の形式の x を入力します。入力値は float_complex に変換されます。
u,(u),(u,v) 形式以外が入力された場合は、is.setstate(ios_base::failbit) を呼びます。
リターン値は is です。

ostream& operator<<(ostream& os, const float_complex& x)

x を os に出力します。
出力形式は u,(u) または (u,v) (u は実数部、v は虚数部) です。
リターン値は os です。

float real(const float_complex& x)

実数部を求めます。
リターン値は x.real() です。

float imag(const float_complex& x)

虚数部を求めます。
リターン値は x.imag() です。

float abs(const float_complex& x)

絶対値を求めます。
リターン値は、$(|x.real()|^2 + |x.imag()|^2)^{1/2}$ です。

float arg(const float_complex& x)

位相角度を求めます。
リターン値は、atan2f(x.imag() , x.real()) です。

float norm(const float_complex& x)

2 乗の絶対値を求めます。
リターン値は、$|x.real()|^2 + |x.imag()|^2$ です。

float_complex conj(const float_complex& x)

共役複素数を求めます。
リターン値は、float_complex(x.real(), (-1)*x.imag()) です。

float_complex polar(const float& rho, const float& theta)

大きさが rho で位相角度 (偏角) が theta の複素数に対応する float_complex 値を求めます。
リターン値は、float_complex(rho*cosf(theta), rho*sinf(theta)) です。

float_complex cos(const float_complex& x)

複素余弦を求めます。
リターン値は、float_complex(cosf(x.real())*coshf(x.imag()), (-1)*sinf(x.real())*sinhf(x.imag())) です。

float_complex cosh(const float_complex& x)

複素双曲余弦を求めます。
リターン値は、cos(float_complex((-1)*x.imag(), x.real())) です。

float_complex exp(const float_complex& x)

指数関数を求めます。
リターン値は、expf(x.real())*cosf(x.imag()),expf(x.real())*sinf(x.imag()) です。

float_complex log(const float_complex& x)

(e を底とする) 自然対数を求めます。
リターン値は、float_complex(logf(abs(x)), arg(x)) です。

float_complex log10(const float_complex& x)

(10 を底とする) 常用対数を求めます。
リターン値は、float_complex(log10f(abs(x)), arg(x)/logf(10)) です。

float_complex pow(const float_complex& x, int y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は次のとおりです。
float_complex pow(const float_complex& x,const float_complex& y) の場合：exp(y*logf(x))
上記以外：exp(y*log(x))

float_complex pow(const float_complex& x, const float& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は次のとおりです。
　float_complex pow(const float_complex& x,const float_complex& y) の場合：exp(y*logf(x))
　上記以外：exp(y*log(x))

float_complex pow(const float_complex& x, const float_complex& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は次のとおりです。
　float_complex pow(const float_complex& x,const float_complex& y) の場合：exp(y*logf(x))
　上記以外：exp(y*log(x))

float_complex pow(const float& x, const float_complex& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は次のとおりです。
　float_complex pow(const float_complex& x,const float_complex& y) の場合：exp(y*logf(x))
　上記以外：exp(y*log(x))

float_complex sin(const float_complex& x)

複素正弦を求めます。
リターン値は、float_complex(sinf(x.real())*coshf(x.imag()), cosf(x.real())*sinhf(x.imag())) です。

float_complex sinh(const float_complex& x)

複素双曲正弦を求めます。
リターン値は、float_complex(0,-1)*sin(float_complex((-1)*x.imag(),x.real())) です。

float_complex sqrt(const float_complex& x)

右半空間における範囲での平方根を求めます。
リターン値は、float_complex(sqrtf(abs(x))*cosf(arg(x)/2), sqrtf(abs(x))*sinf(arg(x)/2)) です。

float_complex tan(const float_complex& x)

複素正接を求めます。
リターン値は、sin(x)/cos(x) です。

float_complex tanh(const float_complex& x)

複素双曲正接を求めます。
リターン値は、sinh(x)/cosh(x) です。

(c) double_complex クラス

種別	定義名	説明
型	value_type	double 型です。
変数	_re	double 精度の実数部を定義します。
	_im	double 精度の虚数部を定義します。
関数	double_complex(double re = 0.0, double im = 0.0)	コンストラクタです。
	double_complex(const float_complex&)	
	double real() const	実数部を求めます。
	double imag() const	虚数部を求めます。
	double_complex& operator=(double rhs)	rhs を実数部にコピーします。虚数部は 0.0 を設定します。
	double_complex& operator+=(double rhs)	rhs を実数部に加算し、和を *this に格納します。
	double_complex& operator-=(double rhs)	rhs を実数部から減算し、差を *this に格納します。
	double_complex& operator*=(double rhs)	rhs を乗算し、積を *this に格納します。
	double_complex& operator/=(double rhs)	rhs で除算し、商を *this に格納します。
	double_complex& operator=(const double_complex& rhs)	rhs をコピーします。
	double_complex& operator+=(const double_complex& rhs)	rhs を加算し、和を *this に格納します。
	double_complex& operator-=(const double_complex& rhs)	rhs を減算し、差を *this に格納します。
	double_complex& operator*=(const double_complex& rhs)	rhs を乗算し、積を *this に格納します。
	double_complex& operator/=(const double_complex& rhs)	rhs で除算し、商を *this に格納します。

double_complex::double_complex(double re = 0.0, double im = 0.0)

クラス double_complex のコンストラクタです。
以下の値で初期化します。
_re = re;
_im = im;

double_complex::double_complex(const float_complex&)

クラス double_complex のコンストラクタです。
以下の値で初期化します。
_re = (double)rhs.real();
_im = (double)rhs.imag();

double double_complex::real() const

実数部を求めます。
リターン値は、this->_re です。

double double_complex::imag() const

虚数部を求めます。
リターン値は、this->_im です。

double_complex& double_complex::operator=(double rhs)

rhs を実数部 (_re) にコピーします。虚数部 (_im) は 0.0 を設定します。
リターン値は *this です。

double_complex& double_complex::operator+=(double rhs)

rhs を実数部 (_re) に加算し、結果を実数部 (_re) に格納します。虚数部 (_im) の値は変わりません。
リターン値は *this です。

double_complex& double_complex::operator-=(double rhs)

rhs を実数部 (_re) から減算し、結果を実数部 (_re) に格納します。虚数部 (_im) の値は変わりません。
リターン値は *this です。

double_complex& double_complex::operator*=(double rhs)

rhs と乗算し、結果を *this に格納します。
(_re=_re*rhs, _im=_im*rhs)
リターン値は *this です。

double_complex& double_complex::operator/=(double rhs)

rhs で除算し、結果を *this に格納します。
(_re=_re/rhs, _im=_im/rhs)
リターン値は *this です。

double_complex& double_complex::operator=(const double_complex& rhs)

rhs をコピーします。
リターン値は *this です。

double_complex& double_complex::operator+=(const double_complex& rhs)

rhs を加算し、結果を *this に格納します。
リターン値は *this です。

double_complex& double_complex::operator-=(const double_complex& rhs)

rhs を減算し、結果を *this に格納します。
リターン値は *this です。

double_complex& double_complex::operator*=(const double_complex& rhs)

rhs と乗算し、結果を *this に格納します。
リターン値は *this です。

double_complex& double_complex::operator/=(const double_complex& rhs)

rhs で除算し、結果を *this に格納します。
リターン値は *this です。

(d) double_complex メンバ外関数

種別	定義名	説明
関数	double_complex operator+(const double_complex& lhs)	lhs の単項＋演算を行います。
	double_complex operator+(const double_complex& lhs, const double_complex& rhs)	lhs と rhs を加算し、和を lhs に格納します。
	double_complex operator+(const double_complex& lhs, const double& rhs)	
	double_complex operator+(const double& lhs, const double_complex& rhs)	
	double_complex operator-(const double_complex& lhs)	lhs の単項－演算を行います。
	double_complex operator-(const double_complex& lhs, const double_complex& rhs)	lhs から rhs を減算し、差を lhs に格納します。
	double_complex operator-(const double_complex& lhs, const double& rhs)	
	double_complex operator-(const double& lhs, const double_complex& rhs)	
	double_complex operator*(const double_complex& lhs, const double_complex& rhs)	lhs と rhs を乗算し、積を lhs に格納します。
	double_complex operator*(const double_complex& lhs, const double& rhs)	
	double_complex operator*(const double& lhs, const double_complex& rhs)	
	double_complex operator/(const double_complex& lhs, const double_complex& rhs)	lhs を rhs で除算し、商を lhs に格納します。
	double_complex operator/(const double_complex& lhs, const double& rhs)	
	double_complex operator/(const double& lhs, const double_complex& rhs)	

種別	定義名	説明
関数	bool operator==(const double_complex& lhs, const double_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator==(const double_complex& lhs, const double& rhs)	
	bool operator==(const double& lhs, const double_complex& rhs)	
	bool operator!=(const double_complex& lhs, const double_complex& rhs)	lhs と rhs の実数部どうし、虚数部どうしを比較します。
	bool operator!=(const double_complex& lhs, const double& rhs)	
	bool operator!=(const double& lhs, const double_complex& rhs)	
	istream& operator>>(istream& is, double_complex& x)	u,(u) または (u,v) (u:実数部、v:虚数部) 形式の x を入力します。
	ostream& operator<<(ostream& os, const double_complex& x)	x を u,(u) または (u,v) (u:実数部、v:虚数部) 形式で出力します。
	double real(const double_complex& x)	実数部を求めます。
	double imag(const double_complex& x)	虚数部を求めます。
	double abs(const double_comlex& x)	絶対値を求めます。
	double arg(const double_complex& x)	位相角度を求めます。
	double norm(const double_complex& x)	２乗の絶対値を求めます。
	double_complex conj(const double_complex& x)	共役複素数を求めます。
	double_complex polar(const double& rho, const double& theta)	大きさが rho で位相角度が theta の複素数に対応する double_complex 値を求めます。
	double_complex cos(const double_complex& x)	複素余弦を求めます。
	double_complex cosh(const double_complex& x)	複素双曲余弦を求めます。
	double_complex exp(const double_complex&)	指数関数を求めます。
	double_complex log(const double_complex& x)	自然対数を求めます。
	double_complex log10(const double_complex& x)	常用対数を求めます。

種別	定義名	説明
関数	double_complex pow(const double _complex& x, int y)	x の y 乗を求めます。
	double_complex pow(const double _complex& x, const double & y)	
	double_complex pow(const double _complex& x, const double _complex& y)	
	double_complex pow(const double & x, const double _complex& y)	x の y 乗を求めます。
	double_complex sin(const double_complex& x)	複素正弦を求めます。
	double_complex sinh(const double_complex& x)	複素双曲正弦を求めます。
	double_complex sqrt(const double_complex& x)	右半空間における範囲での平方根を求めます。
	double_complex tan(const double_complex& x)	複素正接を求めます。
	double_complex tanh(const double_complex& x)	複素双曲正接を求めます。

```
double_complex operator+(const double_complex& lhs)
```

lhs の単項＋演算を行います。
リターン値は lhs です。

```
double_complex operator+(const double_complex& lhs, const double_complex& rhs)
```

lhs と rhs を加算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)+=rhs です。

```
double_complex operator+(const double_complex& lhs, const double& rhs)
```

lhs と rhs を加算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)+=rhs です。

```
double_complex operator+(const double& lhs, const double_complex& rhs)
```

lhs と rhs を加算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)+=rhs です。

```
double_complex operator-(const double_complex& lhs)
```

lhs の単項－演算を行います。
リターン値は、double_complex(-lhs.real(), -lhs.imag()) です。

double_complex operator-(const double_complex& lhs, const double_complex& rhs)

lhs から rhs を減算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)-=rhs です。

double_complex operator-(const double_complex& lhs, const double& rhs)

lhs から rhs を減算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)-=rhs です。

double_complex operator-(const double& lhs, const double_complex& rhs)

lhs から rhs を減算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)-=rhs です。

double_complex operator*(const double_complex& lhs, const double_complex& rhs)

lhs と rhs を乗算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)*=rhs です。

double_complex operator*(const double_complex& lhs, const double& rhs)

lhs と rhs を乗算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)*=rhs です。

double_complex operator*(const double& lhs, const double_complex& rhs)

lhs と rhs を乗算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)*=rhs です。

double_complex operator/(const double_complex& lhs, const double_complex& rhs)

lhs を rhs で除算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)/=rhs です。

double_complex operator/(const double_complex& lhs, const double& rhs)

lhs を rhs で除算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)/=rhs です。

double_complex operator/(const double& lhs, const double_complex& rhs)

lhs を rhs で除算し、結果を lhs に格納します。
リターン値は、double_complex(lhs)/=rhs です。

bool operator==(const double_complex& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator==(const double_complex& lhs, const double& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator==(const double& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()==rhs.real() && lhs.imag()==rhs.imag() です。

bool operator!=(const double_complex& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

bool operator!=(const double_complex& lhs, const double& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

bool operator!=(const double& lhs, const double_complex& rhs)

lhs と rhs の実数部どうし、虚数部どうしを比較します。double 型引数の場合、虚数部は double 型の 0.0 と仮定されます。
リターン値は、lhs.real()!=rhs.real() || lhs.imag()!=rhs.imag() です。

istream& operator>>(istream& is, double_complex& x)

u,(u) または (u,v) (u は実数部、v は虚数部) の形式の複素数 x を入力します。入力値は double_complex に変換されます。
u,(u),(u,v) 形式以外が入力された場合は、is.setstate(ios_base::failbit) を呼びます。
リターン値は is です。

ostream& operator<<(ostream& os, const double_complex& x)

x を os に出力します。
出力形式は u,(u) または (u,v) (u は実数部、v は虚数部) です。
リターン値は os です。

double real(const double_complex& x)

実数部を求めます。
リターン値は x.real() です。

double imag(const double_complex& x)

虚数部を求めます。
リターン値は x.imag() です。

double abs(const double_complex& x)

絶対値を求めます。
リターン値は、$(|x.real()|^2 + |x.imag()|^2)^{1/2}$ です。

double arg(const double_complex& x)

位相角度を求めます。
リターン値は、atan2(x.imag() , x.real()) です。

double norm(const double_complex& x)

2 乗の絶対値を求めます。
リターン値は、$|x.real()|^2 + |x.imag()|^2$ です。

double_complex conj(const double_complex& x)

共役複素数を求めます。
リターン値は、double_complex(x.real(), (-1)*x.imag()) です。

double_complex polar(const double& rho, const double& theta)

大きさが rho で位相角度 (偏角) が theta の複素数に対応する double_complex 値を求めます。
リターン値は、double_complex(rho*cos(theta), rho*sin(theta)) です。

double_complex cos(const double_complex& x)

複素余弦を求めます。
リターン値は、double_complex(cos(x.real())*cosh(x.imag()), (-1)*sin(x.real())*sinh(x.imag())) です。

double_complex cosh(const double_complex& x)

複素双曲余弦を求めます。
リターン値は、cos(double_complex((-1)*x.imag(), x.real())) です。

double_complex exp(const double_complex& x)

指数関数を求めます。
リターン値は、exp(x.real())*cos(x.imag()),exp(x.real())*sin(x.imag()) です。

double_complex log(const double_complex& x)

(e を底とする) 自然対数を求めます。
リターン値は、double_complex(log(abs(x)), arg(x)) です。

double_complex log10(const double_complex& x)

(10 を底とする) 常用対数を求めます。
リターン値は、double_complex(log10(abs(x)), arg(x)/log(10)) です。

double_complex pow(const double_complex& x, int y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は、exp(y*log(x)) です。

double_complex pow(const double_complex& x, const double& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は、exp(y*log(x)) です。

double_complex pow(const double_complex& x, const double_complex& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は、exp(y*log(x)) です。

double_complex pow(const double& x, const double_complex& y)

x の y 乗を求めます。
pow(0,0) のとき、定義域エラーになります。
リターン値は、exp(y*log(x)) です。

double_complex sin(const double_complex& x)

複素正弦を求めます。
リターン値は、double_complex(sin(x.real())*cosh(x.imag()), cos(x.real())*sinh(x.imag())) です。

double_complex sinh(const double_complex& x)

複素双曲正弦を求めます。
リターン値は、double_complex(0,-1)*sin(double_complex((-1)*x.imag(),x.real())) です。

double_complex sqrt(const double_complex& x)

右半空間における範囲での平方根を求めます。
リターン値は、double_complex(sqrt(abs(x))*cos(arg(x)/2), sqrt(abs(x))*sin(arg(x)/2)) です。

double_complex tan(const double_complex& x)

複素正接を求めます。
リターン値は、sin(x)/cos(x) です。

double_complex tanh(const double_complex& x)

複素双曲正接を求めます。
リターン値は、sinh(x)/cosh(x) です。

7.5.4 文字列操作用クラスライブラリ

文字列操作用クラスライブラリに対応するヘッダファイルは以下の通りです。

- <string>
  string クラスを定義します。

本クラスには派生関係はありません。

(a) string クラス

種別	定義名	説明
型	iterator	char* 型です。
	const_iterator	const char* 型です。
定数	npos	文字列の最大長 (UINT_MAX 文字) です。
変数	s_ptr	オブジェクトが文字列を格納している領域へのポインタです。
	s_len	オブジェクトが格納している文字列長です。
	s_res	オブジェクトが文字列を格納するために確保している領域のサイズです。

種別	定義名	説明
関数	string(void)	コンストラクタです。
	string(const string& str, size_t pos = 0, size_t n = npos)	
	string(const char* str, size_t n)	
	string(const char* str)	
	string(size_t n, char c)	
	~string()	デストラクタです。
	string& operator=(const string& str)	str を代入します。
	string& operator=(const char* str)	
	string& operator=(char c)	c を代入します。
	iterator begin()	文字列の先頭ポインタを求めます。
	const_iterator begin() const	
	iterator end()	文字列の最後尾ポインタを求めます。
	const_iterator end() const	
	size_t size() const	格納されている文字列の文字列長を求めます。
	size_t length() const	
	size_t max_size() const	確保している領域のサイズを求めます。
	void resize(size_t n, char c)	格納可能な文字列の文字数を n に変更します。
	void resize(size_t n)	格納可能な文字列の文字数を n に変更します。
	size_t capacity() const	確保している領域のサイズを求めます。
	void reserve(size_t res_arg = 0)	領域の再割り当てを行います。
	void clear()	格納されている文字列を clear します。
	bool empty() const	格納している文字列の文字数が 0 かチェックします。
	const char& operator[](size_t pos) const	s_ptr[pos] を参照します。
	char& operator[](size_t pos)	
	const char& at(size_t pos) const	
	char& at(size_t pos)	
	string& operator+=(const string& str)	str の文字列を追加します。
	string& operator+=(const char* str)	
	string& operator+=(char c)	c の文字を追加します。
	string& append(const string& str)	str の文字列を追加します。
	string& append(const char* str)	

種別	定義名	説明
関数	string& append(const string& str, size_t pos, size_t n)	オブジェクトの位置 pos に str の文字列を n 文字分追加します。
	string& append(const char* str, size_t n)	文字列 str の n 文字分を追加します。
	string& append(size_t n, char c)	n 個の文字 c を追加します。
	string& assign(const string& str)	str の文字列を代入します。
	string& assign(const char* str)	
	string& assign(const string& str, size_t pos, size_t n)	位置 pos に文字列 str の n 文字分を代入します。
	string& assign(const char* str, size_t n)	文字列 str の n 文字分を代入します。
	string& assign(size_t n, char c)	n 個の文字 c を代入します。
	string& insert(size_t pos1, const string& str)	位置 pos1 に str の文字列を挿入します。
	string& insert(size_t pos1, const string& str, size_t pos2, size_t n)	位置 pos1 に str の文字列の位置 pos2 から n 文字分を挿入します。
	string& insert(size_t pos, const char* str, size_t n)	pos の位置に文字列 str を n 文字分挿入します。
	string& insert(size_t pos, const char* str)	pos の位置に文字列 str を挿入します。
	string& insert(size_t pos, size_t n, char c)	位置 pos に n 個の文字 c の文字列を挿入します。
	iterator insert(iterator p, char c = char())	p が指す文字列の前に文字 c を挿入します。
	void insert(iterator p, size_t n, char c)	p が指す文字の前に、n 個の文字 c を挿入します。
	string& erase(size_t pos = 0, size_t n = npos)	位置 pos から n 個分取り除きます。
	iterator erase(iterator position)	position により参照された文字を取り除きます。
	iterator erase(iterator first, iterator last)	範囲 [first, last] において文字を取り除きます。
	string& replace(size_t pos1, size_t n1, const string& str)	位置 pos1 から n1 文字分の文字列を、str の文字列で置き換えます。
	string& replace(size_t pos1, size_t n1, const char* str)	
	string& replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)	位置 pos1 から n1 文字分の文字列を、str の位置 pos2 から n2 文字分の文字列で置き換えます。

種別	定義名	説明
関数	string& replace(size_t pos, size_t n1, const char* str, size_t n2)	位置 pos から n1 文字分の文字列を、n2 個の str の文字列で置き換えます。
	string& replace(size_t pos, size_t n1, size_t n2, char c)	位置 pos から n1 文字分の文字列を、n2 個の文字 c で置き換えます。
	string& replace(iterator i1, iterator i2, const string& str)	位置 i1 から i2 までの文字列を str の文字列で置き換えます。
	string& replace(iterator i1, iterator i2, const char* str)	
	string& replace(iterator i1, iterator i2, const char* str, size_t n)	位置 i1 から i2 までの文字列を str の文字列の n 文字分で置き換えます。
	string& replace(iterator i1, iterator i2, size_t n, char c)	位置 i1 から i2 までの文字列を n 個の文字 c で置き換えます。
	size_t copy(char* str, size_t n, size_t pos = 0) const	位置 pos に文字列 str の n 文字分の文字列をコピーします。
	void swap(string& str)	str の文字列と交換します。
	const char* c_str() const	文字列を格納している領域へのポインタを参照します。
	const char* data() const	
	size_t find(const string& str, size_t pos = 0) const	位置 pos 以降で str の文字列と同じ文字列が最初に現れる位置を検索します。
	size_t find(const char* str, size_t pos = 0) const	
	size_t find(const char* str, size_t pos, size_t n) const	位置 pos 以降で str の n 文字分と同じ文字列が最初に現れる位置を検索します。
	size_t find(char c, size_t pos = 0) const	位置 pos 以降で文字 c が最初に現れる位置を検索します。
	size_t rfind(const string& str, size_t pos = npos) const	位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。

種別	定義名	説明
関数	size_t rfind(const char* str, size_t pos = npos) const	位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。
	size_t rfind(const char* str, size_t pos, size_t n) const	位置 pos 以前で str の n 文字分と同じ文字列が最後に現れる位置を検索します。
	size_t rfind(char c, size_t pos = npos) const	位置 pos 以前で文字 c が最後に現れる位置を検索します。
	size_t find_first_of(const string& str, size_t pos = 0) const	位置 pos 以降で文字列 str に含まれる任意の文字が最初に現れる位置を検索します。
	size_t find_first_of(const char* str, size_t pos = 0) const	
	size_t find_first_of(const char* str, size_t pos, size_t n) const	位置 pos 以降で文字列 str の n 文字分に含まれる任意の文字が最初に現れる位置を検索します。
	size_t find_first_of(char c, size_t pos = 0) const	位置 pos 以降で文字 c が最初に現れる位置を検索します。
	size_t find_last_of(const string& str, size_t pos = npos) const	位置 pos 以前で文字列 str に含まれる任意の文字が最後に現れる位置を検索します。
	size_t find_last_of(const char* str, size_t pos = npos) const	
	size_t find_last_of(const char* str, size_t pos, size_t n) const	位置 pos 以前で文字列 str の n 文字分に含まれる任意の文字が最後に現れる位置を検索します。
	size_t find_last_of(char c, size_t pos = npos) const	位置 pos 以前で文字 c が最後に現れる位置を検索します。
	size_t find_first_not_of(const string& str, size_t pos = 0) const	位置 pos 以降で str 中の任意の文字と異なった文字が最初に現れる位置を検索します。
	size_t find_first_not_of(const char* str, size_t pos = 0) const	
	size_t find_first_not_of(const char* str, size_t pos, size_t n)	位置 pos 以降で str の先頭から n 文字までの任意の文字と異なった文字が最初に現れる位置を検索します。
	size_t find_first_not_of(char c, size_t pos = 0) const	位置 pos 以降で文字 c と異なった文字が最初に現れる位置を検索します。
	size_t find_last_not_of(const string& str, size_t pos = npos) const	位置 pos 以前で str 中の任意の文字と異なった文字が最後に現れる位置を検索します。

種別	定義名	説明
関数	size_t find_last_not_of(const char* str, size_t pos = npos) const	
	size_t find_last_not_of(const char* str, size_t pos, size_t n) const	位置 pos 以前で str の先頭から n 文字までの任意の文字と異なった文字が最後に現れる位置を検索します。
	size_t find_last_not_of(char c, size_t pos = npos) const	位置 pos 以前で文字 c と異なった文字が最後に現れる位置を検索します。
	string substr(size_t pos = 0, size_t n = npos) const	格納された文字列に対し、範囲 [pos,n] の文字列を持つオブジェクトを生成します。
	int compare(const string& str) const	文字列と str の文字列を比較します。
	int compare(size_t pos1, size_t n1, const string& str) const	位置 pos1 から n1 文字分の文字列と str を比較します。
	int compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const	位置 pos1 から n1 文字分の文字列と str の位置 pos2 から n2 文字分の文字列を比較します。
	int compare(const char* str) const	str と比較します。
	int compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const	位置 pos1 から n1 文字分の文字列と str の n2 文字分の文字列を比較します。

string::string(void)

以下のように設定します。
s_ptr=0;
s_len=0;
s_res=1;

string::string(const string& str, size_t pos = 0, size_t n = npos)

str をコピーします。ただし、s_len は、n と s_len の小さい方の値になります。

string::string(const char* str, size_t n)

以下に設定します。
s_ptr=str;
s_len=n;
s_res=n+1;

string::string(const char* str)

以下に設定します。
s_ptr=str;
s_len=str の文字列長 ;
s_res=str の文字列長 +1;

string::string(size_t n, char c)

以下に設定します。
s_ptr= 文字数 n で文字 c の文字列 ;
s_len=n;
s_res=n+1;

string::~string()

クラス string のデストラクタです。
文字列を格納している領域を解放します。

string& string::operator=(const string& str)

str のデータを代入します。
リターン値は *this です。

string& string::operator=(const char* str)

str から string オブジェクトを生成し、そのデータを代入します。
リターン値は *this です。

string& string::operator=(char c)

c から string オブジェクトを生成し、そのデータを代入します。
リターン値は *this です。

string::iterator string::begin()

文字列の先頭ポインタを求めます。
リターン値は、文字列の先頭ポインタです。

string::const_iterator string::begin() const

文字列の先頭ポインタを求めます。
リターン値は、文字列の先頭ポインタです。

string::iterator string::end()

文字列の最後尾ポインタを求めます。
リターン値は、文字列の最後尾ポインタです。

string::const_iterator string::end() const

文字列の最後尾ポインタを求めます。
リターン値は、文字列の最後尾ポインタです。

size_t string::size() const

格納されている文字列の文字列長を求めます。
リターン値は、格納されている文字列の文字列長です。

size_t string::length() const

格納されている文字列の文字列長を求めます。
リターン値は、格納されている文字列の文字列長です。

size_t string::max_size() const

確保している領域のサイズを求めます。
リターン値は、確保している領域のサイズです。

void string::resize(size_t n, char c)

オブジェクトが格納可能な文字列の文字数を n に変更します。
n<=size() のとき、長さを n にした元の文字列と置き換えます。
n>size() のとき、元の文字列の後ろに長さ n になるまで c をつめた文字列と置き換えます。
n<=max_size() である必要があります。
n>max_size() の場合 §n=max_size() として計算します。

void string::resize(size_t n)

オブジェクトが格納可能な文字列の文字数を n に変更します。
n<=size() のとき、長さを n にしたもとの文字列と置き換えます。
n<=max_size() である必要があります。

size_t string::capacity() const

確保している領域のサイズを求めます。
リターン値は、確保している領域のサイズです。

void string::reserve(size_t res_arg = 0)

記憶領域の再割り当てを行います。
reserve() 後、capacity() は reserve() の引数より大きいかまたは等しくなります。
再割り当てを行うと、すべての参照、ポインタ、この数列の中の要素の参照する iterator を無効にします。

void string::clear()

格納されている文字列をクリアします。

bool string::empty() const

格納している文字列の文字数が 0 かチェックします。
リターン値は次のとおりです。
　格納している文字列長が 0 の場合：true
　格納している文字列長が 0 以外の場合：false

const char& string::operator[](size_t pos) const

s_ptr[pos] を参照します。
リターン値は次のとおりです。
n<s_len の場合：s_ptr [pos]
n>=s_len の場合：'\0'

char& string::operator[](size_t pos)

s_ptr[pos] を参照します。
リターン値は次のとおりです。
n<s_len の場合：s_ptr [pos]
n>=s_len の場合：'\0'

const char& string::at(size_t pos) const

s_ptr[pos] を参照します。
リターン値は次のとおりです。
n<s_len の場合：s_ptr [pos]
n>=s_len の場合：'\0'

char& string::at(size_t pos)

s_ptr[pos] を参照します。
リターン値は次のとおりです。
n<s_len の場合：s_ptr [pos]
n>=s_len の場合：'\0'

string& string::operator+=(const string& str)

str が格納している文字列を追加します。
リターン値は *this です。

string& string::operator+=(const char* str)

str から string オブジェクトを生成し、その文字列を追加します。
リターン値は *this です。

string& string::operator+=(char c)

c から string オブジェクトを生成し、その文字列を追加します。
リターン値は *this です。

string& string::append(const string& str)

str の文字列をオブジェクトに追加します。
リターン値は *this です。

string& string::append(const char* str)

str の文字列をオブジェクトに追加します。
リターン値は *this です。

string& string::append(const string& str, size_t pos, size_t n)

オブジェクトの位置 pos に str の文字列を n 文字分追加します。
リターン値は *this です。

string& string::append(const char* str, size_t n)

文字列 str の n 文字分を追加します。
リターン値は *this です。

string& string::append(size_t n, char c)

n 個の文字 c を追加します。
リターン値は *this です。

string& string::assign(const string& str)

str の文字列を代入します。
リターン値は *this です。

string& string::assign(const char* str)

str の文字列を代入します。
リターン値は *this です。

string& string::assign(const string& str, size_t pos, size_t n)

位置 pos に文字列 str の n 文字分を代入します。
リターン値は *this です。

string& string::assign(const char* str, size_t n)

文字列 str の n 文字分を代入します。
リターン値は *this です。

string& string::assign(size_t n, char c)

n 個の文字 c を代入します。
リターン値は *this です。

string& string::insert(size_t pos1, const string& str)

位置 pos1 に str の文字列を挿入します。
リターン値は *this です。

string& string::insert(size_t pos1, const string& str, size_t pos2, size_t n)

位置 pos1 に str の文字列の位置 pos2 から n 文字分を挿入します。
リターン値は *this です。

```
string& string::insert(size_t pos, const char* str, size_t n)
```

pos の位置に文字列 str を n 文字分挿入します。
リターン値は *this です。

```
string& string::insert(size_t pos, const char* str)
```

pos の位置に文字列 str を挿入します。
リターン値は *this です。

```
string& string::insert(size_t pos, size_t n, char c)
```

位置 pos に n 個の文字 c の文字列を挿入します。
リターン値は *this です。

```
string::iterator string::insert(iterator p, char c = char())
```

p が指す文字列の前に、文字 c を挿入します。
リターン値は、挿入された文字です。

```
void string::insert(iterator p, size_t n, char c)
```

p が指す文字の前に、n 個の文字 c を挿入します。

```
string& string::erase(size_t pos = 0, size_t n = npos)
```

位置 pos から n 個分取り除きます。
リターン値は *this です。

```
iterator string::erase(iterator position)
```

position により参照された文字を取り除きます。
リターン値は次のとおりです。
削除要素の次の iterator がある場合：削除要素の次の iterator
削除要素の次の iterator がない場合：end()

```
iterator string::erase(iterator first, iterator last)
```

範囲 [first, last] において文字を取り除きます。
リターン値は次のとおりです。
last の次の iterator がある場合：last の次の iterator
last の次の iterator がない場合：end()

```
string& string::replace(size_t pos1, size_t n1, const string& str)
```

位置 pos1 から n1 文字分の文字列を、str の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(size_t pos1, size_t n1, const char* str)
```

位置 pos1 から n1 文字分の文字列を、str の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2)
```

位置 pos1 から n1 文字分の文字列を、str の位置 pos2 から n2 文字分の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(size_t pos, size_t n1, const char* str, size_t n2)
```

位置 pos から n1 文字分の文字列を、n2 個の str の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(size_t pos, size_t n1, size_t n2, char c)
```

位置 pos から n1 文字分の文字列を、n2 個の文字 c で置き換えます。
リターン値は *this です。

```
string& string::replace(iterator i1, iterator i2, const string& str)
```

位置 i1 から i2 までの文字列を str の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(iterator i1, iterator i2, const char* str)
```

位置 i1 から i2 までの文字列を str の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(iterator i1, iterator i2, const char* str, size_t n)
```

位置 i1 から i2 までの文字列を、str の n 文字分の文字列で置き換えます。
リターン値は *this です。

```
string& string::replace(iterator i1, iterator i2, size_t n, char c)
```

位置 i1 から i2 までの文字列を、n 個の文字 c で置き換えます。
リターン値は *this です。

```
size_t string::copy(char* str, size_t n, size_t pos = 0) const
```

位置 pos に文字列 str の n 文字分の文字列をコピーします。
リターン値は rlen です。

```
void string::swap(string& str)
```

str の文字列と交換します。

```
const char* string::c_str() const
```

文字列を格納している領域へのポインタを参照します。
リターン値は s_ptr です。

```
const char* string::data() const
```

文字列を格納している領域へのポインタを参照します。
リターン値は s_ptr です。

size_t string::find(const string& str, size_t pos = 0) const

位置 pos 以降で str の文字列と同じ文字列が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find(const char* str, size_t pos = 0) const

位置 pos 以降で str の文字列と同じ文字列が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find(const char* str, size_t pos, size_t n) const

位置 pos 以降で str の n 文字分と同じ文字列が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find(char c, size_t pos = 0) const

位置 pos 以降で文字 c が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::rfind(const string& str, size_t pos = npos) const

位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::rfind(const char* str, size_t pos = npos) const

位置 pos 以前で str の文字列と同じ文字列が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::rfind(const char* str, size_t pos, size_t n) const

位置 pos 以前で文字列 str の n 文字分と同じ文字列が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::rfind(char c, size_t pos = npos) const

位置 pos 以前で文字 c が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_of(const string& str, size_t pos = 0) const

位置 pos 以降で文字列 str に含まれる任意の文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_of(const char* str, size_t pos = 0) const

位置 pos 以降で文字列 str に含まれる任意の文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_of(const char* str, size_t pos, size_t n) const

位置 pos 以降で文字列 str の n 文字分に含まれる任意の文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_of(char c, size_t pos = 0) const

位置 pos 以降で文字 c が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_of(const string& str, size_t pos = npos) const

位置 pos 以前で文字列 str に含まれる任意の文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_of(const char* str, size_t pos = npos) const

位置 pos 以前で文字列 str に含まれる任意の文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_of(const char* str, size_t pos, size_t n) const

位置 pos 以前で文字列 str の n 文字分に含まれる任意の文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_of(char c, size_t pos = npos) const

位置 pos 以前で文字 c が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(const string& str, size_t pos = 0) const

位置 pos 以降で str 中の任意の文字と異なった文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(const char* str, size_t pos = 0) const

位置 pos 以降で str 中の任意の文字と異なった文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(const char* str, size_t pos, size_t n) const

位置 pos 以降で str の先頭から n 文字までの任意の文字と異なった文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_first_not_of(char c, size_t pos = 0) const

位置 pos 以降で文字 c と異なった文字が最初に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(const string& str, size_t pos = npos) const

位置 pos 以前で str 中の任意の文字と異なった文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(const char* str, size_t pos = npos) const

位置 pos 以前で str 中の任意の文字と異なった文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(const char* str, size_t pos, size_t n) const

位置 pos 以前で str の先頭から n 文字までの任意の文字と異なった文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

size_t string::find_last_not_of(char c, size_t pos = npos) const

位置 pos 以前で文字 c と異なった文字が最後に現れる位置を検索します。
リターン値は、文字列のオフセットです。

string string::substr(size_t pos = 0, size_t n = npos) const

格納された文字列に対し、範囲 [pos,n] の文字列を持つオブジェクトを生成します。
リターン値は、範囲 [pos,n] の文字列を持つオブジェクトです。

int string::compare(const string& str) const

文字列と str の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：0
　文字列が異なる場合：this->s_len>str.s_len のとき 1
　　this->s_len < str.s_len のとき -1

int string::compare(size_t pos1, size_t n1, const string& str) const

位置 pos1 から n1 文字分の文字列と str を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：0
　文字列が異なる場合：this->s_len>str.s_len のとき 1
　　this->s_len < str.s_len のとき -1

int string::compare(size_t pos1, size_t n1, const string& str, size_t pos2, size_t n2) const

位置 pos1 から n1 文字分の文字列と str の位置 pos2 から n2 文字分の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：0
　文字列が異なる場合：this->s_len>str.s_len のとき 1
　　this->s_len < str.s_len のとき -1

```
int string::compare(const char* str) const
```

str と比較します。
リターン値は次のとおりです。
　文字列が同一の場合：0
　文字列が異なる場合：this->s_len>str.s_len のとき 1
　　this->s_len < str.s_len のとき -1

```
int string::compare(size_t pos1, size_t n1, const char* str, size_t n2 = npos) const
```

位置 pos1 から n1 文字分の文字列と str の n2 文字分の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：0
　文字列が異なる場合：this->s_len>str.s_len のとき 1
　　this->s_len < str.s_len のとき -1

(b) string クラスマニピュレータ

種別	定義名	説明
関数	string operator+(const string& lhs, const string& rhs)	lhsの文字列(または文字)にrhsの文字列(または文字) を追加し、オブジェクトを生成してその文字列を格納します。
	string operator+(const char* lhs, const string& rhs)	
	string operator+(char lhs, const string& rhs)	
	string operator+(const string& lhs, const char* rhs)	
	string operator+(const string& lhs, char rhs)	
	bool operator==(const string& lhs, const string& rhs)	lhs の文字列と rhs の文字列を比較します。
	bool operator==(const char* lhs, const string& rhs)	
	bool operator==(const string& lhs, const char* rhs)	
	bool operator!=(const string& lhs, const string& rhs)	lhs の文字列と rhs の文字列を比較します。
	bool operator!=(const char* lhs, const string& rhs)	
	bool operator!=(const string& lhs, const char* rhs)	
	bool operator<(const string& lhs, const string& rhs)	lhsの文字列長とrhsの文字列長を比較します。
	bool operator<(const char* lhs, const string& rhs)	
	bool operator<(const string& lhs, const char* rhs)	
	bool operator>(const string& lhs, const string& rhs)	lhsの文字列長とrhsの文字列長を比較します。
	bool operator>(const char* lhs, const string& rhs)	
	bool operator>(const string& lhs, const char* rhs)	

種別	定義名	説明
関数	bool operator<=(const string& lhs, const string& rhs)	lhsの文字列長とrhsの文字列長を比較します。
	bool operator<=(const char* lhs, const string& rhs)	
	bool operator<=(const string& lhs, const char* rhs)	
	bool operator>=(const string& lhs, const string& rhs)	lhsの文字列長とrhsの文字列長を比較します。
	bool operator>=(const char* lhs, const string& rhs)	
	bool operator>=(const string& lhs, const char* rhs)	
	void swap(string& lhs, string& rhs)	lhs の文字列と rhs の文字列を交換します。
	istream& operator>>(istream& is, string& str)	文字列を str に抽出します。
	ostream& operator<<(ostream& os, const string& str)	文字列を挿入します。
	istream& getline(istream& is, string& str, char delim)	is から文字列を抽出し、str に付加します。途中で文字 'delim' を検出したら、入力を終了します。
	istream& getline(istream& is, string& str)	is から文字列を抽出し、str に付加します。途中で改行文字を検出したら、入力を終了します。

string operator+(const string& lhs, const string& rhs)

lhs の文字列 (または文字) に rhs の文字列 (または文字) を追加し、オブジェクトを生成してその文字列を格納します。
リターン値は、結合した文字列を格納するオブジェクトです。

string operator+(const char* lhs, const string& rhs)

lhs の文字列 (または文字) に rhs の文字列 (または文字) を追加し、オブジェクトを生成してその文字列を格納します。
リターン値は、結合した文字列を格納するオブジェクトです。

string operator+(char lhs, const string& rhs)

lhs の文字列 (または文字) に rhs の文字列 (または文字) を追加し、オブジェクトを生成してその文字列を格納します。
リターン値は、結合した文字列を格納するオブジェクトです。

string operator+(const string& lhs, const char* rhs)

lhs の文字列 (または文字) に rhs の文字列 (または文字) を追加し、オブジェクトを生成してその文字列を格納します。
リターン値は、結合した文字列を格納するオブジェクトです。

```
string operator+(const string& lhs, char rhs)
```

lhs の文字列 (または文字) に rhs の文字列 (または文字) を追加し、オブジェクトを生成してその文字列を格納します。
リターン値は、結合した文字列を格納するオブジェクトです。

```
bool operator==(const string& lhs, const string& rhs)
```

lhs の文字列と rhs の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：true
　文字列が異なる場合：false

```
bool operator==(const char* lhs, const string& rhs)
```

lhs の文字列と rhs の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：false
　文字列が異なる場合：true

```
bool operator==(const string& lhs, const char* rhs)
```

lhs の文字列と rhs の文字列を比較します。
リターン値は次のとおりです。
　文字列が同一の場合：false
　文字列が異なる場合：true

```
bool operator<(const string& lhs, const string& rhs)
```

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
　lhs.s_len < rhs.s_len の場合：true
　lhs.s_len >= rhs.s_len の場合：false

```
bool operator<(const char* lhs, const string& rhs)
```

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
　lhs.s_len < rhs.s_len の場合：true
　lhs.s_len >= rhs.s_len の場合：false

```
bool operator<(const string& lhs, const char* rhs)
```

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
　lhs.s_len < rhs.s_len の場合：true
　lhs.s_len >= rhs.s_len の場合：false

```
bool operator>(const string& lhs, const string& rhs)
```

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
　lhs.s_len > rhs.s_len の場合：true
　lhs.s_len <= rhs.s_len の場合：false

bool operator>(const char* lhs, const string& rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len > rhs.s_len の場合：true
lhs.s_len <= rhs.s_len の場合：false

bool operator>(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len > rhs.s_len の場合：true
lhs.s_len <= rhs.s_len の場合：false

bool operator<=(const string& lhs, const string& rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len <= rhs.s_len の場合：true
lhs.s_len > rhs.s_len の場合：false

bool operator<=(const char* lhs, const string& rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len <= rhs.s_len の場合：true
lhs.s_len > rhs.s_len の場合：false

bool operator<=(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len <= rhs.s_len の場合：true
lhs.s_len > rhs.s_len の場合：false

bool operator>=(const string& lhs, const string& rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len >= rhs.s_len の場合：true
lhs.s_len < rhs.s_len の場合：false

bool operator>=(const char* lhs, const string& rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len >= rhs.s_len の場合：true
lhs.s_len < rhs.s_len の場合：false

bool operator>=(const string& lhs, const char* rhs)

lhs の文字列長と rhs の文字列長を比較します。
リターン値は次のとおりです。
lhs.s_len >= rhs.s_len の場合：true
lhs.s_len < rhs.s_len の場合：false

void swap(string& lhs, string& rhs)

lhs の文字列と rhs の文字列を交換します。

istream& operator>>(istream& is, string& str)

文字列を str に抽出します。
リターン値は is です。

ostream& operator<<(ostream& os, const string& str)

文字列を挿入します。
リターン値は os です。

istream& getline(istream& is, string& str, char delim)

is から文字列を抽出し、str に付加します。
途中で文字 'delim' を検出したら、入力を終了します。
リターン値は is です。

istream& getline(istream& is, string& str)

is から文字列を抽出し、str に付加します。
途中で改行文字を検出したら、入力を終了します。
リターン値は is です。

7.6 未サポートライブラリ

C 言語仕様で定義しているライブラリのうち、本コンパイラでサポートしていないライブラリを表 7.16 に示します。

表 7.16 サポートしていないライブラリ

	ヘッダファイル	ライブラリ名
1	locale.h*1	setlocale、localeconv
2	signal.h*1	signal、raise
3	stdio.h	remove、rename、tmpfile、tmpnam、fgetpos、fsetpos
4	stdlib.h	abort、atexit、exit、_Exit、getenv、system、 mblen、mbtowc、wctomb、mbstowcs、wcstombs
5	string.h	strcoll、strxfrm
6	time.h*1	clock、difftime、mktime、time、asctime、ctime、gmtime、 localtime、strftime
7	wctype.h	iswalnum、iswalpha、iswblank、iswcntrl、iswdigit、iswgraph、 iswlower、iswprint、iswpunct、iswspace、iswupper、iswxdigit、 iswctype、wctype、towlower、towupper、towctrans、wctrans
8	wchar.h	wcsftime、wcscoll、wcsxfrm、wctob、mbrtowc、wcrtomb、 mbsrtowcs, wcsrtombs

注 ヘッダファイルをサポートしません。

8. スタートアップ

この章では、スタートアップルーチンについて説明します。

8.1 概要

C/C++ 言語で書かれたプログラムを実行するには、システムへ組み込むための ROM イメージ作成処理や、ユーザ・プログラム（main 関数）の起動などを行うプログラムが別途必要となります。このプログラムのことをスタートアップ・ルーチンと呼びます。

ユーザが作成したプログラムを実行するには、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。RX 用の統合開発環境では、ユーザがシステムに合わせて変更できるようにスタートアップ・ルーチンのソース・ファイルを提供しています。

8.2 ファイルの構成

RX 用の統合開発環境が提供しているスタートアップ・ルーチンは、以下のとおりです。

表 8.1 統合開発環境で生成されるプログラムの一覧

	ファイル名	内容
(a)	resetprg.c	初期設定ルーチン (リセットベクタ関数)
(b)	intprg.c	ベクタ関数の定義
(c)	vecttbl.c	固定ベクタテーブル *1
(d)	dbsct.c	セクションの初期化処理 (テーブル)
(e)	lowsrc.c	低水準インタフェースルーチン (C 言語部分)
(f)	lowlvl.src	低水準インタフェースルーチン (アセンブリ言語部分)
(g)	sbrk.c	低水準インタフェースルーチン (sbrk 関数)
(h)	typedefine.h	型定義ヘッダ
(i)	vect.h	ベクタ関数のヘッダ
(j)	stacksct.h	スタックサイズの設定
(k)	lowsrc.h	低水準インタフェースルーチン (C 言語ヘッダ)
(l)	sbrk.h	低水準インタフェースルーチン (sbrk 関数のヘッダ)

注 1. RXv1 命令セットアーキテクチャの場合です。
RXv1 命令セットアーキテクチャ以外の場合は、「例外ベクタテーブル」となります。

8.3 スタートアッププログラム

本節では、プログラムの実行に必要な環境を設定するための処理について説明します。ただし、プログラムを実行する環境はユーザシステムごとに異なりますので、ユーザシステムの仕様に合わせて実行環境の設定プログラムを作成する必要があります。

本節の内容は、標準となるスタートアップについて説明しています。PIC/PID 機能におけるアプリケーションに使用するスタートアップは特別な対応が必要ですので「8.5.7 アプリケーションのスタートアップ」も参照ください。

必要な手続きの概要は以下の通りです。

- ベクタテーブルの設定
  パワーオンリセットで初期設定ルーチン (PowerON_Reset_PC) が起動されるように、リセットベクタテーブルを設定します。
- 初期設定
  main 関数に到達するまでに必要な手続きを行います。レジスタやセクションの初期化、各種初期設定ルーチンの呼び出しを行います。

- 低水準インタフェースルーチンの作成
  標準入出力 (stdio.h、ios、streambuf、istream、ostream)、メモリ管理ライブラリ (stdlib.h、new) を使用する場合に必要なライブラリ関数とユーザシステムとの間のインタフェースルーチンです。
- 終了処理ルーチン (exit、atexit、abort)*1 の作成
  プログラムの終了処理を行います。

注 1. プログラムの終了処理を行う C ライブラリ関数 exit、atexit、abort 関数を使用する場合は、ユーザシステムにあわせてこれらの関数を作成する必要があります。
C++ プログラムを使用する場合、または C ライブラリ関数 assert マクロを使用する場合、abort 関数は必ず作成する必要があります。

8.3.1 ベクタテーブルの設定

主に、パワーオンリセットで、初期設定ルーチン PowerON_Reset_PC が呼び出されるようにするために、リセットベクタに PowerON_Reset_PC のアドレスを設定します。以下にそのコーディング例を示します。
リセットベクタ以外の、特権命令例外、アクセス例外、未定義命令例外、浮動小数点例外およびノンマスカブル割り込みの各ベクタは、「固定ベクタテーブル」または「例外ベクタテーブル」に登録する必要があります。登録方法は命令セットアーキテクチャにより異なりますので、詳細については RX ファミリのソフトウェアマニュアルおよびハードウェアマニュアル等を参照してください。

例

```
extern void PowerOn_Reset_PC(void);

#pragma section VECTTBL  /* #pragma section 宣言により RESET_Vectors を     */
                         /* CVECTTBL セクションに出力します。                 */
                         /* リンク時に start オプションで CVECTTBL セクションを */
                         /* リセットベクタに割り付けるよう指定します。          */
void (*const RESET_Vectors[])(void)={
    PowerON_Reset_PC,
};
```

8.3.2 初期設定

初期設定ルーチン PowerON_Reset_PC は、main 関数を実行する前、および実行した後に必要な手続きを記述する関数です。初期設定ルーチンの中で必要となる処理を、順番に記述します。

(1) 初期設定処理向けの PSW レジスタ初期化
初期設定処理を行うための、PSW レジスタ初期化を実施します。たとえば、初期設定処理中、割り込みを受け付けない設定にするために、PSW に対して割り込み禁止の設定をします。
リセット時の PSW は全 bit ゼロに初期化され、割り込み許可ビット (I ビット) も割り込み禁止状態 (ゼロ) として初期化されています。

(2) スタックポインタの初期化
スタックポインタ (USP レジスタおよび ISP レジスタ) を初期化します。PowerON_Reset_PC 関数に対して "#pragma entry" 宣言することにより、コンパイラが自動的に ISP/USP 初期化コードを関数先頭に生成します。
PowerON_Reset_PC 関数は、#pragma entry 宣言されているため、この手続きを記述する必要はありません。

(3) ベースレジスタに使用する汎用レジスタの初期化
コンパイラで base オプションを使用している場合、プログラム全体でベースアドレスとして使用する汎用レジスタを初期化する必要があります。PowerON_Reset_PC 関数に対して "#pragma entry" 宣言することにより、コンパイラが自動的に各レジスタへの初期化コードを関数先頭に生成します。
PowerON_Reset_PC 関数は、#pragma entry 宣言されているため、この手続きを記述する必要はありません。

(4) 各種制御レジスタの初期化
可変ベクタテーブルの配置アドレスを、INTB に書き込みます。その他、必要に応じて、FINTV、FPSW、BPC、BPSW、EXTB、DPSW の初期化を行います。これらの初期化は、コンパイラの組み込み関数を使って行うことができます。
ただし、PSW だけは、割り込みマスク設定を維持するため、初期化はまだ行いません。

(5) 三角関数演算器の初期化
-tfu オプションを指定している場合、組み込み関数 __init_tfu() を呼び出して三角関数演算器を初期化します。

(6) セクションの初期化処理

RAM 領域セクションの初期化用ルーチン (_INITSCT) を呼び出します。未初期化データセクションはゼロ初期化されます。初期化データセクションは、ROM 領域の初期値を RAM 領域へコピーします。_INITSCT は、標準ライブラリとして提供されます。
初期化対象のセクションは、ユーザがセクション初期化用テーブル (DTBL,BTBL) へ記述する必要があります。_INITSCT 関数が使用するセクションの先頭アドレスおよび最終アドレスを、セクションアドレス演算子を用いて設定します。
セクション初期化用テーブルのセクション名は、未初期化データ領域を C$BSEC、初期化データ領域を C$DSEC で宣言します。
以下にコーディング例を示します。

例

```
#pragma section C C$DSEC        // セクション名を C$DSEC にします
extern const struct {
    void *rom_s;                // 初期化データセクションの ROM 上の先頭アドレスメンバ
    void *rom_e;                // 初期化データセクションの ROM 上の最終アドレスメンバ
    void *ram_s;                // 初期化データセクションの RAM 上の先頭アドレスメンバ
} DTBL[] = {__sectop("D"), __secend("D"), __sectop("R")};

#pragma section C C$BSEC        // セクション名を C$BSEC にします
extern const struct {
    void *b_s;                  // 未初期化データセクションの先頭アドレスメンバ
    void *b_e;                  // 未初期化データセクションの最終アドレスメンバ
} BTBL[] = {__sectop("B"), __secend("B")};
```

(7) ライブラリの初期化処理

C/C++ 言語ライブラリ関数使用時に、必要な初期化を実施するルーチン (_INITLIB) を呼び出します。
実際に使用する機能に合わせた必要最低限の初期設定を行うために、以下の指針を参考にしてください。

- 作成した低水準インタフェースルーチンの中で初期設定が必要な場合、低水準インタフェースルーチンの仕様に合わせた初期設定 (_INIT_LOWLEVEL) が必要です。
- rand 関数、strtok 関数を使用する場合、標準入出力以外の初期設定 (_INIT_OTHERLIB) が必要です。

ライブラリの初期設定を行うプログラム例を以下に示します。

```
#include <stdio.h>
#include <stdlib.h>
#define IOSTREAM 3
const size_t _sbrk_size = 1024;    // ヒープ領域確保サイズの最小単位を指定します

extern char *_s1ptr;

#ifdef __cplusplus
extern "C" {
#endif
void _INITLIB (void)
{
    _INIT_LOWLEVEL();              // 低水準インタフェースルーチンの初期設定をします
    _INIT_IOLIB();                 // 入出力ライブラリの初期設定をします
    _INIT_OTHERLIB();              // rand 関数、strtok 関数の初期設定をします
}

void _INIT_LOWLEVEL (void)
{                                  // 低水準ライブラリで必要な初期設定をしてください
}

void _INIT_OTHERLIB(void)
{
    srand(1);                      // rand 関数を使用する場合の初期設定です
    _s1ptr=NULL;                   // strtok 関数を使用する場合の初期設定です
}
#ifdef __cplusplus
}
#endif
```

注 1. 標準入出力ファイルのファイル名を指定します。この名前は、低水準インタフェースルーチン「open」で使用します。

注 2. コンソール等対話的な装置の場合、バッファリングを行わないためのフラグを立てます。

(8) グローバルクラスオブジェクトの初期化
C++ 言語のプログラム開発の場合、グローバルに宣言されたクラスオブジェクトのコンストラクタを呼び出すためのルーチン (_CALL_INIT) を呼び出します。_CALL_INIT は、標準ライブラリとして提供されます。

(9) main 関数実行向けの PSW 初期化
PSW レジスタを初期化します。割り込みマスクの設定も、ここで解除します。

(10) PSW の PM ビットの変更
リセット以降は特権モード (PSW の PM ビットがゼロ) で動作していますが、ユーザモードに切り替えたい場合は、組み込み関数の chg_pmusr を実行します。
chg_pmusr 関数にはいくつか注意事項がありますので、使用する場合は、4.2.6 組み込み関数の chg_pmusr の項目を参照ください。

(11) ユーザプログラムの実行
main 関数を実行します。

(12) グローバルクラスオブジェクトの後処理
C++ 言語のプログラム開発の場合、グローバルに宣言されたクラスオブジェクトのデストラクタを呼び出すためのルーチン (_CALL_END) を呼び出します。_CALL_END は、標準ライブラリとして提供されます。

8.3.3 初期設定ルーチンの記述例

「8.3.2 初期設定」で説明した、PowerON_Reset_PC 関数のコーディング例を示します。
なお、統合開発環境で生成される実際の初期設定ルーチンは、「8.4 コーディング例」を参照ください。

```
#include         <machine.h>
#include         <_h_c_lib.h>
#include         "typedefine.h"
#include         "stacksct.h"

#ifdef __cplusplus
extern "C" {
#endif
void PowerON_Reset_PC(void);
void main(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus                          // Use SIM I/O
extern "C" {
#endif
extern void _INITLIB(void);
#ifdef __cplusplus
}
#endif

#define PSW_init  0x00010000
#define FPSW_DPSW_init 0x00000100

#pragma section ResetPRG
#pragma entry PowerON_Reset_PC
void PowerON_Reset_PC(void)
{
#if (__RX_ISA_VERSION__ >= 2) || defined(__RXV2)
    set_extb(__sectop("EXCEPTVECT"));
#endif
    set_intb(__sectop("C$VECT"));
#ifdef __FPU
    set_fpsw(FPSW_DPSW_init);
#ifdef __DPFPU
    __set_dpsw(FPSW_DPSW_init);
#endif
#endif
#if __TFU == 1
    __init_tfu();
#endif

    _INITSCT();
    _INITLIB();
    set_psw(PSW_init);
    main();
    brk();
}
```

8.3.4 低水準インタフェースルーチン

低水準インタフェースルーチンとは、ユーザシステムの仕様に合わせたライブラリ関数を実現するために、ライブラリ関数から呼び出されるユーザ関数のことです。次の場合に作成する必要があります。

(1) 標準入出力やメモリ管理に関するライブラリ関数を使用する場合
(2) ライブラリ関数がリエントラントである必要がある場合

表 8.2 に C ライブラリ関数で使用している低水準インタフェースルーチンの一覧を示します。

表 8.2 低水準インタフェースルーチンの一覧

No.	名称 *1	機能
1	open	ファイルのオープン
2	close	ファイルのクローズ
3	read	ファイルからの読み込み
4	write	ファイルへの書き出し
5	lseek	ファイルの読み込み／書き出しの位置の設定
6	sbrk	メモリ領域の確保
7	errno_addr *2	errno アドレスの取得
8	wait_sem *2	セマフォの確保
9	signal_sem *2	セマフォの解放

注 1. 関数名 open、close、read、write、lseek、sbrk、error_addr、wait_sem、signal_sem は低水準インタフェースルーチンの予約済み識別子です。ユーザプログラム中では使用しないでください。

注 2. リエントラントライブラリを使用する場合に必要です。

低水準インタフェースルーチンで必要な初期化は、プログラム起動時に行う必要があります。これは、ライブラリ初期設定処理 _INITLIB の中の「_INIT_LOWLEVEL」という関数の中で行ってください。
以下、低水準入出力の基本的な考え方を説明したあと、各インタフェースルーチンの仕様を説明します。

(1) 入出力の考え方
標準入出力ライブラリでは、ファイルを FILE 型のデータによって管理しますが、低水準インタフェースルーチンでは、実際のファイルと 1 対 1 に対応する正の整数を与え、これによって管理します。この整数をファイル番号といいます。
open ルーチンでは、与えられたファイル名に対してファイル番号を与えます。open ルーチンでは、この番号によってファイルの入出力ができるように、以下の情報を設定する必要があります。

- ファイルのデバイスの種類 (コンソール、プリンタ、ディスクファイル等)。
  コンソールやプリンタ等の特殊なデバイスに対しては、特別なファイル名をシステムで決めておいて open ルーチンで判定する必要があります。
- ファイルのバッファリングはバッファの位置、サイズ等の情報。
- ディスクファイルは、ファイルの先頭から次に読み込み / 書き出しを行う位置までのバイトオフセット。

open ルーチンで設定した情報に基づいて、以後、入出力 (read、write ルーチン)、読み込み / 書き出し位置の設定 (lseek ルーチン) を行います。
close ルーチンでは、出力ファイルのバッファリングを行っている場合はバッファの内容を実際のファイルに書き出し、open ルーチンで設定したデータの領域が再使用できるようにしてください。

(2) 低水準インタフェースルーチンの仕様
本項では低水準インタフェースルーチンを作成するための仕様を説明します。以下、各ルーチンごとに、ルーチンを呼び出す際のインタフェースとその動作および実現上の注意事項を示します。
各ルーチンのインタフェースは以下の形式で示します。なお、低水準インタフェースルーチンは必ず関数原型にしてください。また C++ プログラム内で宣言する場合は「extern "C"」を付加してください。

凡例：

（ルーチン名）

［説明］

-（ルーチンの機能概要を示します）

［リターン値］

正常： （正常に終了した場合のリターン値を示します）
異常： （エラーが生じた場合のリターン値を示します）

［引数］

（名前） （意味）
（インタフェースに示す引数名です） （引数として渡される値を示します）

long open(const char *name, long mode, long flg)

[説明]

- 第 1 引数のファイル名に対応するファイルを操作するための準備をします。
- open ルーチンでは、後で読み込み / 書き出しを行うために、ファイルの種類 (コンソール、プリンタ、ディスクファイル等) を決定しなければなりません。ファイルの種類は、以後 open ルーチンで返したファイル番号を用いて読み込み / 書き出しを行うたびに参照する必要があります。
- 第 2 引数の mode は、ファイルをオープンする時の処理の指定です。このデータの各ビットの意味について以下に示します。

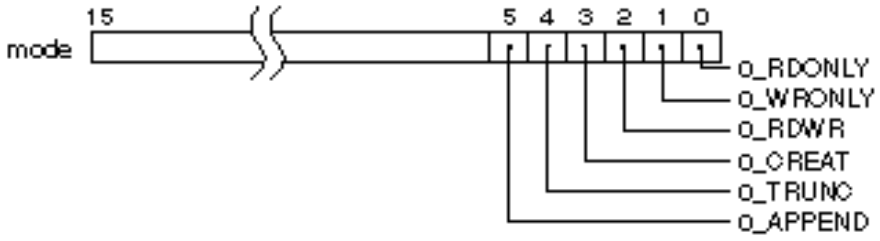


表 8.3 open ルーチン mode ビット説明

mode ビット	説明
O_RDONLY (0 ビット)	ビットが 1 のとき、ファイルを読み込み専用にオープン
O_WRONLY (1 ビット)	ビットが 1 のとき、ファイルを書き出し専用にオープン
O_RDWR (2 ビット)	ビットが 1 のとき、ファイルを読み込み、書き出し両用にオープン
O_CREAT (3 ビット)	ビットが 1 のとき、ファイル名で示すファイルが存在しない場合にファイルを新規に作成
O_TRUNC (4 ビット)	ビットが 1 のとき、ファイル名で示すファイルが存在する場合に ファイルの内容を捨て、ファイルのサイズを 0 に更新
O_APPEND (5 ビット)	次に読み込み / 書き出しを行うファイル内の位置を設定 ビットが 0 のとき：ファイルの先頭に設定 ビットが 1 のとき：ファイルの最後に設定

mode で示したファイルの処理の指定と実際のファイルの性質が矛盾する場合はエラーにしてください。正常にファイルがオープンできた場合は、以後の read、write、lseek、close ルーチンで使用されるファイル番号 **(** 正の整数 **)** を返してください。ファイル番号と実際のファイルの対応は低水準インタフェースルーチンで管理する必要があります。オープンに失敗した場合は **-1** を返してください。

[リターン値]

正常： 正常オープンしたファイルのファイル番号
異常： -1

[引数]

name ファイルのファイル名を指す文字
mode ファイルをオープンするときの処理の指定
flg ファイルをオープンするときの処理の指定 (常に 0777)

long close(long fileno)

［説明］

- open ルーチンで得られたファイル番号が引数として渡されます。
- open ルーチンで設定したファイル管理情報の領域を、再び使用できるように解放してください。また、低水準インタフェースルーチン内で出力ファイルのバッファリングを行っている場合は、バッファの内容を実際のファイルに書き出してください。
- ファイルを正常にクローズできた場合は 0、失敗した場合は -1 を返してください。

［リターン値］

正常：	0
異常：	-1

［引数］

fileno	クローズするファイル番号

long read(long fileno, unsigned char *buf, long count)

［説明］

- 第 1 引数 (fileno) で示すファイルから、第 2 引数 (buf) の指す領域へデータを読み込みます。読み込むデータのバイト数は第 3 引数 (count) で示します。
- ファイルが終了した場合、count で示されたバイト数以下のバイト数しか読み込むことができません。
- ファイルの読み込み / 書き出しの位置は、読み込んだバイト数だけ先に進みます。
- 正常に読み込みができた場合は、実際に読み込んだバイト数を返してください。読み込みに失敗した場合は -1 を返してください。

［リターン値］

正常：	実際に読み込んだバイト数
異常：	-1

［引数］

fileno	読み込みの対象となるファイル番号
buf	読み込んだデータを格納する領域
count	読み込むバイト数

long write(long fileno, const unsigned char *buf, long count)

［説明］

- 第 2 引数 (buf) の指す領域から、第 1 引数 (fileno) の示すファイルにデータを書き出します。書き込むデータのバイト数は第 3 引数 (count) で示します。
- ファイルを書き出そうとしているデバイス (ディスク等) が満杯の時は、count で示されたバイト数以下のバイト数しか書き出すことができません。実際に書き出すことのできたバイト数が何度か連続して 0 バイトの場合、ディスクが満杯であると判断してエラー (-1) を返すように実現することをお勧めします。
- ファイルの読み込み / 書き出しの位置は、書き出したバイト数だけ先に進みます。
- 正常に書き出しができた場合は、実際に書き出したバイト数を返してください。書き出しに失敗した場合は -1 を返してください。

［リターン値］

正常：	実際に書き出されたバイト数
異常：	-1

［引数］

fileno	書き出しの対象となるファイル番号
buf	書き出すデータ領域
count	書き出すバイト数

long lseek(long fileno, long offset, long base)

［説明］

- ファイルの読み込み / 書き出しを行うファイル内の位置を、バイト単位で設定します。
- 新しいファイル内の位置は、第 3 引数 (base) によって、以下の方法で計算し設定してください。
  (1) base が 0 のときファイルの先頭から offset バイトの位置に設定します。
  (2) base が 1 のとき現在の位置に offset バイトを加えた位置に設定します。
  (3) base が 2 のときファイルのサイズに offset バイトを加えた位置に設定します。
- ファイルがコンソールやプリンタ等の対話的なデバイスの場合や、新しいオフセットの値が負になったり、(1)(2) のときファイルのサイズを超える場合はエラーにします。
- 正しくファイル位置を設定できた場合は、新しい読み込み / 書き出し位置のファイルの先頭からのオフセットを、そうでない場合は -1 を返してください。

［リターン値］

正常：	新しいファイルの読み込み / 書き出し位置のファイルの先頭からのオフセット (バイト単位)
異常：	-1

［引数］

fileno	対象となるファイル番号
offset	読み込み / 書き出しの位置を示すオフセット (バイト単位)
base	オフセットの起点

char *sbrk(size_t size)

［説明］

- メモリ領域を割り付けるサイズが引数として渡されます。
- 連続して sbrk ルーチンを呼び出す場合は、下位アドレスから順に連続した領域が割り付けられるようにしてください。割り付けるメモリ領域が不足した場合はエラーにしてください。
- 正常に割り付けができた場合は、割り付けた領域の先頭のアドレスを、割り付けに失敗した場合は、「(char *)-1」を返してください。
- 標準ライブラリ関数 malloc, calloc, realloc、あるいは C++ 関数の new 式のいずれかを用いる場合は、少なくとも 16 バイト以上のメモリ領域が割り付けできるようにしてください。

［リターン値］

正常：	割り付けた領域の先頭アドレス
異常：	(char *)-1

［引数］

size	割り付けるデータのサイズ

long *errno_addr(void)

[説明]

- 現在のタスクが持つエラー番号のアドレスを返却します。
- 標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

[リターン値]

正常：	現在のタスクが持つエラー番号のアドレス

long wait_sem(long semnum)

［説明］

- semnum で示されたセマフォを確保します。
- 確保できた場合は 1、確保できなかった場合は 0 を返してください。
- 標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

［リターン値］

正常：	1
異常：	0

［引数］

semnum	セマフォ ID

long signal_sem(long semnum)

［説明］

- semnum で示されたセマフォを解放します。
- 解放できた場合は 1、解放できなかった場合は 0 を返してください。
- 標準ライブラリ構築ツールで reent オプションを指定して作成した標準ライブラリを使用する場合に、本関数は必要になります。

［リターン値］

正常：	1
異常：	0

［引数］

semnum	セマフォ ID

(3) 低水準インタフェースルーチンコーディング例

```
/************************************************************************/
/*                                  lowsrc.c:                           */
/*----------------------------------------------------------------------*/
/*      RX ファミリ  シミュレータ・デバッガ  インタフェースルーチン                */
/*              - 標準入出力 (stdin,stdout,stderr) だけをサポートしています -      */
/************************************************************************/
#include <string.h>

/*  ファイル番号  */
#define STDIN  0                        /*  標準入力 ( コンソール )*/
#define STDOUT 1                        /*  標準出力 ( コンソール )*/
#define STDERR 2                        /*  標準エラー出力 ( コンソール )*/

#define FLMIN 0                         /*  最小のファイル番号 */
#define FLMAX 3                         /*  ファイル数の最大値 */

/*  ファイルのフラグ  */
#define O_RDONLY 0x0001                 /*  読み込み専用 */
#define O_WRONLY 0x0002                 /*  書き出し専用 */
#define O_RDWR 0x0004                   /*  読み書き両用 */

/*  特殊文字コード  */
#define CR 0x0d                         /*  復帰 */
#define LF 0x0a                         /*  改行 */

/*  sbrk で管理する領域サイズ  */
#define HEAPSIZE 1024

/************************************************************************/
/*                              参照関数の宣言 :                          */
/*     シミュレータ・デバッガでコンソールへの文字入出力を行うアセンブリプログラムの参照    */
/************************************************************************/
extern void charput(char);              /*  一文字入力処理 */
extern char charget(void);              /*  一文字出力処理 */

/************************************************************************/
/*                              静的変数の定義 :                          */
/*                    低水準インタフェースルーチンで使用する静的変数の定義             */
/************************************************************************/
char flmod[FLMAX];                      /*  オープンしたファイルのモード設定場所  */

union HEAP_TYPE{
    long dummy;                         /*  4 バイトアライメントにするためのダミー */
    char heap[HEAPSIZE];                /*  sbrk で管理する領域の宣言 */
};

static union HEAP_TYPE heap_area;

static char *brk=(char*)&heap_area;     /*  sbrk で割り付けた領域の最終アドレス  */
```

```
/*****************************************************************************/
/*                                  open: ファイルのオープン                   */
/*                               リターン値：ファイル番号（成功）              */
/*                                          -1        （失敗）               */
/*****************************************************************************/
long open(const char *name,                         /*  ファイル名 */
          long mode,                                /*  ファイルのモード */
          long flg)                                 /*  処理の指定（未使用）*/
{
    /*  ファイル名に従ってモードをチェックし、ファイル番号を返す  */

    if (strcmp(name,"stdin")==0) {                  /*  標準入力ファイル  */
        if ((mode&O_RDONLY)==0) {
            return (-1);
        }
        flmod[STDIN]=mode;
        return (STDIN);
    }

    else if (strcmp(name,"stdout")==0) {            /*  標準出力ファイル  */
        if ((mode&O_WRONLY)==0) {
            return (-1);
        }
        flmod[STDOUT]=mode;
        return (STDOUT);
    }

    else if (strcmp(name,"stderr")==0){             /* 標準エラー出力ファイル  */
        if ((mode&O_WRONLY)==0) {
            return (-1);
        }
        flmod[STDERR]=mode;
        return (STDERR);
    }

    else {
        return (-1);                                /* エラー  */
    }
}

/*****************************************************************************/
/*                                 close: ファイルのクローズ                  */
/*                              リターン値：0          （成功）               */
/*                                         -1          （失敗）               */
/*****************************************************************************/
long close(long fileno)                             /*  ファイル番号  */
{
    if (fileno<FLMIN || FLMAX<fileno) {             /*  ファイル番号の範囲チェック  */
        return -1;
    }

    flmod[fileno]=0;                                /*  ファイルのモードリセット  */

    return 0;
}
```

```
/*****************************************************************************/
/*                                  read: データの読み込み                   */
/*                          リターン値：実際に読み込んだ文字数（成功）       */
/*                                  -1                （失敗）               */
/*****************************************************************************/
long read(long fileno,                          /*  ファイル番号           */
         unsigned char *buf,                    /*  転送先バッファアドレス  */
         long  count)                           /*  読み込み文字数          */
{
    unsigned long i;

    /*  ファイル名に従ってモードをチェックし、一文字づつ入力してバッファに格納  */

    if (flmod[fileno]&O_RDONLY || flmod[fileno]&O_RDWR) {
        for (i=count;i>0;i--) {
            *buf=charget();
            if (*buf==CR) {                     /*  改行文字の置き換え  */
                *buf=LF;
            }
        buf++;
        }
        return count;
    }

    else {
        return -1;
    }
}

/*****************************************************************************/
/*                                  write: データの書き出し                  */
/*                          リターン値：実際に書き出した文字数（成功）       */
/*                                  -1                （失敗）               */
/*****************************************************************************/
long write(long fileno,                         /*  ファイル番号  */
          const unsigned char *buf,             /*  転送元バッファアドレス  */
          long count)                           /*  書き出し文字数  */
{
    unsigned long i;
    unsigned char c;

    /*  ファイル名に従ってモードをチェックし、一文字づつ出力  */

    if (flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR) {
        for (i=count; i>0; i--) {
            c=*buf++;
            charput(c);
        }
        return count;
    }

    else {
        return -1;
    }
}
```

```
/************************************************************************/
/*                        lseek: ファイルの読み込み／書き出し位置の設定                */
/*          リターン値：読み込み／書き出し位置のファイル先頭からのオフセット  ( 成功 )    */
/*                     -1                                    ( 失敗 )    */
/*                   ( コンソール入出力では、lseek はサポートしていません )             */
/************************************************************************/
long lseek(long fileno,                     /*  ファイル番号           */
           long offset,                     /*  読み込み／書き出し位置   */
           long  base)                      /*  オフセットの起点         */
{
    return -1;
}

/************************************************************************/
/*                               sbrk: メモリ領域の割り付け                    */
/*                      リターン値：割り付けた領域の先頭アドレス  ( 成功 )             */
/*                               -1                          ( 失敗 )             */
/************************************************************************/
char *sbrk(size_t size)                     /*  割り付ける領域のサイズ  */
{
    char *p;

    /*  空き領域のチェック  */

    if (brk+size>heap_area.heap+HEAPSIZE) {
        return (char *)-1;
    }

    p=brk;                                  /* 領域の割り付け        */
    brk+=size;                              /* 最終アドレスの更新     */
    return p;
}
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           lowlvl.src                                          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  RX Family Simulator/Debugger Interface Routine                              ;
;             - Inputs and outputs one character -                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          .GLB    _charput
          .GLB    _charget

SIM_IO    .EQU 0h

          .SECTION   P,CODE
;--------------------------------------------------------------------------
;  _charput:
;--------------------------------------------------------------------------
_charput:
          MOV.L      #IO_BUF,R2
          MOV.B      R1,[R2]
          MOV.L      #1220000h,R1
          MOV.L      #PARM,R3
          MOV.L      R2,[R3]
          MOV.L      R3,R2
          MOV.L      #SIM_IO,R3
          JSR        R3
          RTS

;--------------------------------------------------------------------------
;  _charget:
;--------------------------------------------------------------------------
_charget:
         MOV.L       #1210000h,R1
         MOV.L       #IO_BUF,R2
         MOV.L       #PARM,R3
         MOV.L       R2,[R3]
         MOV.L       R3,R2
         MOV.L       #SIM_IO,R3
         JSR         R3
         MOV.L       #IO_BUF,R2
         MOVU.B      [R2],R1
         RTS

;--------------------------------------------------------------------------
;  I/O Buffer
;--------------------------------------------------------------------------
          .SECTION  B,DATA,ALIGN=4
PARM:     .BLKL     1
          .SECTION  B_1,DATA
IO_BUF:   .BLKB     1
          .END
```

(4) リエントラントライブラリ用低水準インタフェースルーチン例
リエントラントライブラリ用低水準インタフェース例を示します。ライブラリジェネレータで reent オプションを指定して作成したライブラリを使用する場合に必要になります。
wait_sem 関数、signal_sem 関数でセマフォの確保に失敗した場合、errno に以下を設定し、ライブラリ関数からリターンします。

表 8.4 リエントラントライブラリが errno に設定するエラー番号

関数名	エラー番号（マクロ名）	意味
wait_sem	EMALRESM	malloc 用セマフォ資源確保に失敗しました
	ETOKRESM	strtok 用セマフォ資源確保に失敗しました
	EFLSRESM	_Files 用セマフォ資源確保に失敗しました
	EMBLNRESM	mbrlen 用セマフォ資源確保に失敗しました
signal_sem	EMALFRSM	malloc 用セマフォ資源解放に失敗しました
	ETOKFRSM	strtok 用セマフォ資源解放に失敗しました
	EFLSFRSM	_Files 用セマフォ資源解放に失敗しました
	EMBLNFRSM	mbrlen 用セマフォ資源解放に失敗しました

割り込みに関しては、セマフォ確保後により優先度の高い割り込みが発生し、セマフォ確保を行うとデッドロックが発生するため、リソースを共有するような処理が割り込みでネストしないようにしてください。

```
#define MALLOC_SEM   1  /* Semaphore No. for malloc */
#define STRTOK_SEM   2  /* Semaphore No. for strtok */
#define FILE_TBL_SEM 3  /* Semaphore No. for fopen  */
#define MBRLEN_SEM   4  /* Semaphore No. for mbrlen */
#define FPSWREG_SEM  5  /* Semaphore No. for FPSW register */
#define FILES_SEM    6  /* Semaphore No. for _Files */
#define SEMSIZE     26  /* FILES_SEM + _nfiles (assumed _nfiles=20) */

#define TRUE 1
#define FALSE 0
#define OK 1
#define NG 0
extern long *errno_addr(void);
extern long wait_sem(long);
extern long signal_sem(long);
static long sem_errno;
static int force_fail_signal_sem = FALSE;
static int semaphore[SEMSIZE];

/*************************************************************************/
/*                  errno_addr: Acquisition of errno address             */
/*                        Return value: errno address                    */
/*************************************************************************/
long *errno_addr(void)
{
    /* Return the errno address of the current task */
    return (&sem_errno);
}

/*************************************************************************/
/*           wait_sem: Defines the specified numbers of semaphores       */
/*                       Return value: OK(=1) (Normal)                   */
/*                                     NG(=0) (Error)                    */
/*************************************************************************/
long wait_sem(long semnum) /* Semaphore ID */
{
    if((0 < semnum) && (semnum < SEMSIZE)) {
        if(semaphore[semnum] == FALSE) {
            semaphore[semnum] = TRUE;
            return(OK);
        }
    }
    return(NG);
}

/*************************************************************************/
/*          signal_sem: Releases the specified numbers of semaphores     */
/*                       Return value: OK(=1) (Normal)                   */
/*                                     NG(=0) (Error)                    */
/*************************************************************************/
long signal_sem(long semnum) /* Semaphore ID */
{
    if(!force_fail_signal_sem) {
        if((0 <= semnum) && (semnum < SEMSIZE)) {
            if( semaphore[semnum] == TRUE ) {
                semaphore[semnum] = FALSE;
                return(OK);
            }
        }
    }
    return(NG);
}
```

8.3.5 終了処理ルーチン

(1) 終了処理の登録と実行 (atexit) ルーチンの作成例
終了処理の登録を行うライブラリ atexit 関数の作成法を示します。
atexit 関数では、引数として渡された関数のアドレスを、終了処理のテーブルに登録します。登録された関数の個数が限界値 (ここでは、登録できる個数を 32 個とします) を超えた場合、あるいは同じ関数が二度以上登録された場合はリターン値として 0 以外 (ここでは 1) を返します。そうでなければ 0 を返します。
以下にプログラム例を示します。

例

```
#include <stdlib.h>

long _atexit_count=0 ;

void (*_atexit_buf[32])(void) ;

#ifdef __cplusplus
extern "C"
#endif
long atexit(void (*f)(void))
{
    int i;

    for(i=0; i<_atexit_count ; i++)        // 既に登録されていないかチェックします
        if(_atexit_buf[i]==f)
            return 1;
    if (_atexit_count==32)                 // 登録数の限界値をチェックします
        return 1;
    else {
        _atexit_buf[_atexit_count++]=f;    // 関数のアドレスを登録します
        return 0;
    }
}
```

(2) プログラムの終了 (exit) ルーチンの作成例
プログラムの終了処理を行うライブラリ exit 関数の作成法を示します。プログラムの終了処理は、ユーザシステムによって異なりますので、以下のプログラム例を参考に、ユーザシステムの仕様に従った終了処理を作成してください。
exit 関数は、引数として渡されたプログラムの終了コードに従ってプログラムの終了処理を行い、プログラム起動時の環境に戻ります。ここでは、終了コードを外部変数に設定して、main 関数を呼び出す直前に setjmp 関数で退避した環境に戻ることによって実現します。プログラム実行前の環境に戻るためには、次の関数「callmain」を作成し、初期設定関数「PowerON_Reset_PC」から関数「main」を呼び出す代わりに、関数「callmain」を呼び出してください。
以下にプログラム例を示します。

```
#include <setjmp.h>
#include <stddef.h>

extern long _atexit_count ;
extern void (*_atexit_buf[32])(void) ;
#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
int main(void);
extern jmp_buf _init_env ;
int _exit_code ;

#ifdef __cplusplus
extern "C"
#endif
void exit(int code)
{
    int i;
    _exit_code=code ;                          // _exit_code にリターンコードを設定します
    for(i=_atexit_count-1; i>=0; i--)            // atexit 関数で登録した関数を順次実行し
ます
        (*_atexit_buf[i])();
    _CLOSEALL();                                 // オープンした関数を全てクローズします
    longjmp(_init_env, 1) ;                     // setjmp で退避した環境にリターンします
}
#ifdef __cplusplus
extern "C"
#endif
void callmain(void)
{
     // setjmp を用いて現在の環境を退避し、main 関数を呼び出します
     if(!setjmp(_init_env))
         _exit_code=main();                      // exit 関数からのリターン時には処理を終
了します
}
```

(3) 異常終了 (abort) ルーチンの作成例

異常終了の場合は、ご使用になっているユーザシステムの仕様に従って、プログラムを異常終了させる処理を行ってください。

C++ プログラムを使用する場合、以下のときにも abort 関数を呼び出します。

- 例外処理が正しく動作しなかった場合
- 純粋仮想関数自体をコールした場合
- dynamic_cast に失敗した場合
- typeid に失敗した場合
- クラス配列の delete 時に情報が取れなかった場合
- クラスオブジェクトのデストラクタコール情報登録時に矛盾が発生した場合

以下、標準出力装置にメッセージを出力したあと、ファイルをクローズしてから無限ループしてリセットを待つプログラム例を示します。

例

```
#include <stdio.h>
#ifdef __cplusplus
extern "C"
#endif
void _CLOSEALL(void);
#ifdef __cplusplus
extern "C"
#endif
void abort(void)
{
        printf("program is abort !!\n");     // メッセージを出力します
        _CLOSEALL();                         // ファイルをクローズします
        while(1)    ;                        // 無限ループします
}
```

8.4 コーディング例

統合開発環境で生成される実際のスタートアッププログラムの例として、CPU 種別として RX610 を選択したシミュレータ用の場合の例を示します。

(1) ソースファイル
スタートアッププログラムは、表 8.5 に示すファイルから構成されます。

表 8.5 統合開発環境で生成されるプログラムの一覧

	ファイル名	内容
(a)	resetprg.c	初期設定ルーチン (リセットベクタ関数)
(b)	intprg.c	ベクタ関数の定義
(c)	vecttbl.c	固定ベクタテーブル *1
(d)	dbsct.c	セクションの初期化処理 (テーブル)
(e)	lowsrc.c	低水準インタフェースルーチン (C 言語部分)
(f)	lowlvl.src	低水準インタフェースルーチン (アセンブリ言語部分)
(g)	sbrk.c	低水準インタフェースルーチン (sbrk 関数)
(h)	typedefine.h	型定義ヘッダ
(i)	vect.h	ベクタ関数のヘッダ
(j)	stacksct.h	スタックサイズの設定
(k)	lowsrc.h	低水準インタフェースルーチン (C 言語ヘッダ)
(l)	sbrk.h	低水準インタフェースルーチン (sbrk 関数のヘッダ)

注 1. RXv1 命令セットアーキテクチャの場合です。
RXv1 命令セットアーキテクチャ以外の場合は、「例外ベクタテーブル」となります。

ファイル内容を、以下、（a）～（l）に示します。

(a) resetprg.c -- 初期設定ルーチン (リセットベクタ関数)

```
#include  <machine.h>
#include  <_h_c_lib.h>
//#include   <stddef.h>           // Remove the comment when you use errno
//#include   <stdlib.h>           // Remove the comment when you use rand()
#include  "typedefine.h"         // Define Types
#include  "stacksct.h"           // Stack Sizes (Interrupt and User)

#ifdef __cplusplus               // For Use Reset vector
extern "C" {
#endif
void PowerON_Reset_PC(void);
void main(void);
#ifdef __cplusplus
}
#endif

#ifdef __cplusplus               // For Use SIM I/O
extern "C" {
#endif
extern void _INIT_IOLIB(void);
extern void _CLOSEALL(void);
#ifdef __cplusplus
}
#endif

#define PSW_init       0x00010000      // PSW bit pattern
#define FPSW_DPSW_init 0x00000000      // FPSW/DPSW bit base pattern

//extern void srand(_UINT);       // Remove the comment when you use rand()
//extern _SBYTE *_s1ptr;          // Remove the comment when you use strtok()

//#ifdef __cplusplus              // Use Hardware Setup
//extern "C" {
//#endif
//extern void HardwareSetup(void);
//#ifdef __cplusplus
//}
//#endif

//#ifdef __cplusplus             // Remove the comment when you use global class
object
//extern "C" {                    // Sections C$INIT and C$END will be generated
//#endif
//extern void _CALL_INIT(void);
//extern void _CALL_END(void);
//#ifdef __cplusplus
//}
//#endif
#pragma section ResetPRG         // output PowerON_Reset_PC to PResetPRG section

#pragma entry PowerON_Reset_PC

void PowerON_Reset_PC(void)
{
#if (__RX_ISA_VERSION__ >= 2) || defined(__RXV2)
    set_extb(__sectop("EXCEPTVECT")); // Remove the comment when you want to set
an address
                                     // into the Exception Vector Table Register
of RXv2 or later
```

```
#endif
     set_intb(__sectop("C$VECT"));
#ifdef __FPU
#ifdef __ROZ                    // Initialize FPSW/DPSW
#define _ROUND 0x00000001       // Let FPSW/DPSW RM/DRM bits=01 (round to zero)
#else
#define _ROUND 0x00000000     // Let FPSW/DPSW RM/DRM bits=00 (round to nearest)
#endif
#ifdef __DOFF
#define _DENOM 0x00000100       // Let FPSW/DPSW DN/DDN bit=1 (denormal as zero)
#else
#define _DENOM 0x00000000       // Let FPSW/DPSW DN/DDN bit=0 (denormal as is)
#endif
     set_fpsw(FPSW_DPSW_init | _ROUND | _DENOM);
#ifdef __DPFPU
     __set_dpsw(FPSW_DPSW_init | _ROUND | _DENOM);
#endif
#endif

     _INITSCT();                  // Initialize Sections

     _INIT_IOLIB();               // Use SIM I/O

//   errno=0;                     // Remove the comment when you use errno
//   srand((_UINT)1);             // Remove the comment when you use rand()
//   _s1ptr=NULL;                 // Remove the comment when you use strtok()

//   HardwareSetup();             // Use Hardware Setup

//   _CALL_INIT();               // Remove the comment when you use global class
object

     set_psw(PSW_init);           // Set Ubit & Ibit for PSW
//   chg_pmusr();                 // Remove the comment when you need to change
PSW PMbit (SuperVisor->User)

     main();

     _CLOSEALL();                 // Use SIM I/O

//   _CALL_END();                // Remove the comment when you use global class
object

     brk();
}
```

(b) intprg.c -- ベクタ関数の定義

```
#include <machine.h>
#include "vect.h"
#pragma section IntPRG

// Exception(Supervisor Instruction)
void Excep_SuperVisorInst(void){/* brk(); */}

// Exception(Undefined Instruction)
void Excep_UndefinedInst(void){/* brk(); */}

// Exception(Floating Point)
void Excep_FloatingPoint(void){/* brk(); */}

// NMI
void NonMaskableInterrupt(void){/* brk(); */}

// Dummy
void Dummy(void){/* brk(); */}

// BRK
void Excep_BRK(void){ wait(); }
```

(c) vecttbl.c -- 固定ベクタテーブル

```
#include "vect.h"

#pragma section C FIXEDVECT

void (*const Fixed_Vectors[])(void) = {
//;0xffffffd0  Exception(Supervisor Instruction)
    Excep_SuperVisorInst,
//;0xffffffd4  Reserved
    Dummy,
//;0xffffffd8  Reserved
    Dummy,
//;0xffffffdc  Exception(Undefined Instruction)
    Excep_UndefinedInst,
//;0xffffffe0  Reserved
    Dummy,
//;0xffffffe4  Exception(Floating Point)
    Excep_FloatingPoint,
//;0xffffffe8  Reserved
    Dummy,
//;0xffffffec  Reserved
    Dummy,
//;0xfffffff0  Reserved
    Dummy,
//;0xfffffff4  Reserved
    Dummy,
//;0xfffffff8  NMI
    NonMaskableInterrupt,
//;0xfffffffc  RESET
//;<<VECTOR DATA START (POWER ON RESET)>>
//;Power On Reset PC
PowerON_Reset_PC
//;<<VECTOR DATA END (POWER ON RESET)>>
};
```

【ご参考】
RXv1 命令セットアーキテクチャ以外を選択した場合 （例外ベクタテーブル）

```
#include "vect.h"

#pragma section C EXCEPTVECT

void (*const Excep_Vectors[])(void) = {
//;0xffffff80  Reserved
    Dummy,
//;0xffffff84  Reserved
    Dummy,
//;0xffffff88  Reserved
    Dummy,
//;0xffffff8c  Reserved
    Dummy,
//;0xffffff90  Reserved
    Dummy,
//;0xffffff94  Reserved
    Dummy,
//;0xffffff98  Reserved
    Dummy,
//;0xffffff9c  Reserved
    Dummy,
//;0xffffffa0  Reserved
    Dummy,
//;0xffffffa4  Reserved
    Dummy,
//;0xffffffa8  Reserved
    Dummy,
//;0xffffffac  Reserved
    Dummy,
//;0xffffffb0  Reserved
    Dummy,
//;0xffffffb4  Reserved
    Dummy,
//;0xffffffb8  Reserved
    Dummy,
//;0xffffffbc  Reserved
    Dummy,
//;0xffffffc0  Reserved
    Dummy,
//;0xffffffc4  Reserved
    Dummy,
//;0xffffffc8  Reserved
    Dummy,
//;0xffffffcc  Reserved
    Dummy,
//;0xffffffd0  Exception(Supervisor Instruction)
    Excep_SuperVisorInst,
//;0xffffffd4  Exception(Access Instruction)
    Excep_AccessInst,
//;0xffffffd8  Reserved
    Dummy,
//;0xffffffdc  Exception(Undefined Instruction)
    Excep_UndefinedInst,
//;0xffffffe0  Reserved
    Dummy,
//;0xffffffe4  Exception(Floating Point)
    Excep_FloatingPoint,
```

```
//;0xffffffe8  Reserved
    Dummy,
//;0xfffffffc  Reserved
    Dummy,
//;0xffffffff0  Reserved
    Dummy,
//;0xfffffff4  Reserved
    Dummy,
//;0xfffffff8  NMI
    NonMaskableInterrupt,
};

#pragma section C RESETVECT

void (*const Reset_Vectors[])(void) = {
//;0xfffffffc  RESET
    PowerON_Reset_PC
};
```

(d) dbsct.c -- セクションの初期化処理 (テーブル)

```
#include "typedefine.h"

#pragma unpack

#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s;       /* Start address of the initialized data section in ROM
*/
    _UBYTE *rom_e;       /* End address of the initialized data section in ROM
*/
    _UBYTE *ram_s;       /* Start address of the initialized data section in RAM
*/
}   _DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") }
};
#pragma section C C$BSEC
extern const struct {
    _UBYTE *b_s;         /* Start address of non-initialized data section */
    _UBYTE *b_e;         /* End address of non-initialized data section */
}   _BTBL[] = {
    { __sectop("B"), __secend("B") },
    { __sectop("B_2"), __secend("B_2") },
    { __sectop("B_1"), __secend("B_1") }
};

#pragma section

/*
** CTBL prevents excessive output of W0561100 messages when linking.
** Even if CTBL is deleted, the operation of the program does not change.
*/
_UBYTE * const _CTBL[] = {
    __sectop("C_1"), __sectop("C_2"), __sectop("C"),
    __sectop("W_1"), __sectop("W_2"), __sectop("W")
};

#pragma packoption
```

(e) lowsrc.c -- 低水準インタフェースルーチン (C 言語部分)

```
#include <string.h>
#include <stdio.h>
#include <stddef.h>
#include "lowsrc.h"

#define STDIN  0
#define STDOUT 1
#define STDERR 2

#define FLMIN  0
#define _MOPENR   0x1
#define _MOPENW   0x2
#define _MOPENA   0x4
#define _MTRUNC   0x8
#define _MCREAT   0x10
#define _MBIN     0x20
#define _MEXCL    0x40
#define _MALBUF   0x40
#define _MALFIL   0x80
#define _MEOF     0x100
#define _MERR     0x200
#define _MLBF     0x400
#define _MNBF     0x800
#define _MREAD    0x1000
#define _MWRITE   0x2000
#define _MBYTE    0x4000
#define _MWIDE    0x8000

#define O_RDONLY  0x0001
#define O_WRONLY  0x0002
#define O_RDWR    0x0004
#define O_CREAT   0x0008
#define O_TRUNC   0x0010
#define O_APPEND  0x0020

#define CR 0x0d
#define LF 0x0a

extern const long _nfiles;
char flmod[IOSTREAM];

unsigned char sml_buf[IOSTREAM];

#define FPATH_STDIN     "C:\\stdin"
#define FPATH_STDOUT    "C:\\stdout"
#define FPATH_STDERR    "C:\\stderr"

extern void charput(unsigned char);
extern unsigned char charget(void);

#include <stdio.h>
FILE *_Files[IOSTREAM];
char *env_list[] = {
    "ENV1=temp01",
    "ENV2=temp02",
    "ENV9=end",
    '\0'
};
```

```
char **environ = env_list;

void _INIT_IOLIB( void )
{
    _Files[0] = stdin;
    _Files[1] = stdout;
    _Files[2] = stderr;

    if( freopen( FPATH_STDIN, "r", stdin ) == NULL )
        stdin->_Mode = 0xffff;
    stdin->_Mode  = _MOPENR;
    stdin->_Mode |= _MNBF;
    stdin->_Bend = stdin->_Buf + 1;

    if( freopen( FPATH_STDOUT, "w", stdout ) == NULL )
        stdout->_Mode = 0xffff;
    stdout->_Mode |= _MNBF;
    stdout->_Bend = stdout->_Buf + 1;

    if( freopen( FPATH_STDERR, "w", stderr ) == NULL )
        stderr->_Mode = 0xffff;
    stderr->_Mode |= _MNBF;
    stderr->_Bend = stderr->_Buf + 1;
}

void _CLOSEALL( void )
{
    long i;
    for( i=0; i < _nfiles; i++ )
    {
        if( _Files[i]->_Mode & (_MOPENR | _MOPENW | _MOPENA ) )
        fclose( _Files[i] );
    }
}

long open(const char *name,
     long  mode,
     long  flg)
{

    if( strcmp( name, FPATH_STDIN ) == 0 )
    {
        if( ( mode & O_RDONLY ) == 0 ) return -1;
        flmod[STDIN] = mode;
        return STDIN;
    }
    else if( strcmp( name, FPATH_STDOUT ) == 0 )
    {
        if( ( mode & O_WRONLY ) == 0 ) return -1;
        flmod[STDOUT] = mode;
        return STDOUT;
    }
    else if(strcmp(name, FPATH_STDERR ) == 0 )
    {
        if( ( mode & O_WRONLY ) == 0 ) return -1;
        flmod[STDERR] = mode;
        return STDERR;
    }
    else return -1;
```

```
long close( long fileno )
{
    return 1;
}

long write(long  fileno,
      const unsigned char *buf,
      long  count)
{
    long    i;
    unsigned char    c;

    if(flmod[fileno]&O_WRONLY || flmod[fileno]&O_RDWR)
    {
        if( fileno == STDIN ) return -1;
        else if( (fileno == STDOUT) || (fileno == STDERR) )
        {
            for( i = count; i > 0; --i )
            {
                c = *buf++;
                charput(c);
            }
            return count;
        }
        else return -1;
    }
    else return -1;
}

long read( long fileno, unsigned char *buf, long count )
{
   long i;
       if((flmod[fileno]&_MOPENR) || (flmod[fileno]&O_RDWR)){
             for(i = count; i > 0; i--){
                   *buf = charget();
                   if(*buf==CR){
                         *buf = LF;
                   }
                   buf++;
             }
             return count;
       }
       else {
             return -1;
       }
}

long lseek( long fileno, long offset, long base )
{
    return -1L;
}
```

(f) lowlvl.src -- 低水準インタフェースルーチン (アセンブリ言語部分)

```
          .GLB    _charput
          .GLB    _charget

SIM_IO    .EQU 0h

          .SECTION   P,CODE
;------------------------------------------------------------------------
;  _charput:
;------------------------------------------------------------------------
_charput:
          MOV.L      #IO_BUF,R2
          MOV.B      R1,[R2]
          MOV.L      #1220000h,R1
          MOV.L      #PARM,R3
          MOV.L      R2,[R3]
          MOV.L      R3,R2
          MOV.L      #SIM_IO,R3
          JSR        R3
          RTS

;------------------------------------------------------------------------
;  _charget:
;------------------------------------------------------------------------
_charget:
         MOV.L       #1210000h,R1
         MOV.L       #IO_BUF,R2
         MOV.L       #PARM,R3
         MOV.L       R2,[R3]
         MOV.L       R3,R2
         MOV.L       #SIM_IO,R3
         JSR         R3
         MOV.L       #IO_BUF,R2
         MOVU.B      [R2],R1
         RTS

;------------------------------------------------------------------------
;  I/O Buffer
;------------------------------------------------------------------------
          .SECTION  B,DATA,ALIGN=4
PARM:     .BLKL     1
          .SECTION  B_1,DATA
IO_BUF:   .BLKB     1
          .END
```

(g) sbrk.c -- 低水準インタフェースルーチン (sbrk 関数)

```
#include <stddef.h>
#include <stdio.h>
#include "typedefine.h"
#include "sbrk.h"

_SBYTE  *sbrk(size_t size);

//const size_t _sbrk_size=                /* Specifies the minimum unit of*/
                                          /* the defined heap area*/

extern _SBYTE *_s1ptr;

union HEAP_TYPE {
    _SDWORD  dummy ;                      /* Dummy for 4-byte boundary*/
    _SBYTE heap[HEAPSIZE];                /* Declaration of the area managed by
sbrk*/
};

static union HEAP_TYPE heap_area ;

/* End address allocated by sbrk*/
static _SBYTE *brk=(_SBYTE *)&heap_area;

/************************************************************************/
/*     sbrk:Memory area allocation                                      */
/*          Return value:Start address of allocated area (Pass)         */
/*                       -1                              (Failure)      */
/************************************************************************/
_SBYTE  *sbrk(size_t size)                    /* Assigned area size   */
{
    _SBYTE  *p;

    if(brk+size > heap_area.heap+HEAPSIZE){     /* Empty area size      */
        p = (_SBYTE *)-1;
    }
    else {
        p = brk;                              /* Area assignment      */
        brk += size;                          /* End address update   */
    }
    return p;
}
```

(h) typedefine.h -- 型定義ヘッダ

```
typedef signed char _SBYTE;
typedef unsigned char _UBYTE;
typedef signed short _SWORD;
typedef unsigned short _UWORD;
typedef signed int _SINT;
typedef unsigned int _UINT;
typedef signed long _SDWORD;
typedef unsigned long _UDWORD;
typedef signed long long _SQWORD;
typedef unsigned long long _UQWORD;
```

(i) vect.h -- ベクタ関数のヘッダ

```
// Exception(Supervisor Instruction)
#pragma interrupt (Excep_SuperVisorInst)
void Excep_SuperVisorInst(void);

// Exception(Undefined Instruction)
#pragma interrupt (Excep_UndefinedInst)
void Excep_UndefinedInst(void);

// Exception(Floating Point)
#pragma interrupt (Excep_FloatingPoint)
void Excep_FloatingPoint(void);

// NMI
#pragma interrupt (NonMaskableInterrupt)
void NonMaskableInterrupt(void);

// Dummy
#pragma interrupt (Dummy)
void Dummy(void);

// BRK
#pragma interrupt (Excep_BRK(vect=0))
void Excep_BRK(void);

//;<<VECTOR DATA START (POWER ON RESET)>>
//;Power On Reset PC
extern void PowerON_Reset_PC(void);
//;<<VECTOR DATA END (POWER ON RESET)>>
```

(j) stacksct.h -- スタックサイズの設定

```
// #pragma stacksize su=0x100        // Remove the comment when you use user
stack
#pragma stacksize si=0x300
```

(k) lowsrc.h -- 低水準インタフェースルーチン (C 言語ヘッダ)

```
/*Number of I/O Stream*/
#define IOSTREAM 20
```

(l) sbrk.h -- 低水準インタフェースルーチン (sbrk 関数のヘッダ)

```
/* size of area managed by sbrk */
#define HEAPSIZE 0x400
```

(2) 実行コマンド

これらのファイルをビルドするのに必要なコマンド列の例を示します。
この例では、ユーザプログラム (main 関数を含む) は UserProgram.c、生成するロードモジュールやライブラリなどのファイル名を LoadModule(拡張子を除いた部分) とします。

```
lbgrx -isa=rxv1 -output=LoadModule.lib
ccrx -isa=rxv1 -output=obj UserProgram.c
ccrx -isa=rxv1 -output=obj resetprg.c
ccrx -isa=rxv1 -output=obj intprg.c
ccrx -isa=rxv1 -output=obj vecttbl.c
ccrx -isa=rxv1 -output=obj dbsct.c
ccrx -isa=rxv1 -output=obj lowsrc.c
asrx -isa=rxv1 lowlvl.src
ccrx -isa=rxv1 -output=obj sbrk.c
rlink -rom=D=R,D_1=R_1,D_2=R_2 -list=LoadModule.map
-start=B_1,R_1,B_2,R_2,B,R,SI/01000,PResetPRG/
0FFFF8000,C_1,C_2,C,C$*,D_1,D_2,D,P,PIntPRG,
W*,L/0FFFF8100,FIXEDVECT/0FFFFFFD0 -=LoadModule.lib -output=LoadModule.abs
UserProgram.obj resetprg.obj intprg.obj vecttbl.obj dbsct.obj lowsrc.obj
lowlvl.obj sbrk.obj
rlink -output=LoadModule.sty -form=stype -output=LoadModule.mot LoadModule.abs
```

【ご参考】
RXv2 命令アーキテクチャを選択した場合のコマンド列の例を示します。

```
lbgrx -isa=rxv2 -output=LoadModule.lib
ccrx -isa=rxv2 -output=obj UserProgram.c
ccrx -isa=rxv2 -output=obj resetprg.c
ccrx -isa=rxv2 -output=obj intprg.c
ccrx -isa=rxv2 -output=obj vecttbl.c
ccrx -isa=rxv2 -output=obj dbsct.c
ccrx -isa=rxv2 -output=obj lowsrc.c
asrx -isa=rxv2 lowlvl.src
ccrx -isa=rxv2 -output=obj sbrk.c
rlink -rom=D=R,D_1=R_1,D_2=R_2 -list=LoadModule.map -
start=B_1,R_1,B_2,R_2,B,R,SU,SI/04,PResetPRG/
0FFFF8000,C_1,C_2,C,C$DSEC,C$BSEC,C$INIT,C$VTBL,C$VECT,D_1,D_2,D,P,PIntPRG,W_1,W_
2,W,L/0FFFF8100,EXCEPTVECT/0FFFFFF80,RESETVECT/0FFFFFFFC -=LoadModule.lib -
output=LoadModule.abs UserProgram.obj resetprg.obj intprg.obj vecttbl.obj
dbsct.obj lowsrc.obj lowlvl.obj sbrk.obj
rlink -output=LoadModule.sty -form=stype -output=LoadModule.mot LoadModule.abs
```

8.5 PIC/PID 機能の利用

本章では、PIC/PID 機能の概要と、PIC/PID 機能を利用する場合のスタートアップの作成方法について説明します。
PIC/PID 機能は、一度リンクが完了して配置アドレスが確定した ROM 上のコードやデータを、リンクをやり直すことなく、任意のアドレスに配置して利用できるようにする機能です。
PIC は位置独立コード (Position Independent Code)、PID は位置独立データ (Position Independent Data) をそれぞれ意味します。PIC を生成する機能が PIC 機能、PID を生成する機能が PID 機能で、ここでは、これらの機能を総称して PIC/PID 機能と呼びます。

8.5.1 用語の定義

(1) マスタとアプリケーション
PIC/PID 機能では、ROM 上のコードやデータを PIC や PID にしたプログラムをアプリケーション、アプリケーションを実行させるのに必要なプログラムをマスタと呼びます。
マスタは、アプリケーションの起動処理のほか、アプリケーションから呼び出される共有ライブラリ、およびアプリケーションの RAM 領域を持ちます。PIC および PID はアプリケーションにのみ含まれ、マスタには含まれていません。

(2) 共有ライブラリ
マスタ内にあり、複数のアプリケーションから呼び出すことのできる関数群です。

(3) ジャンプテーブル
アプリケーションから共有ライブラリ関数への呼び出しを中継するプログラムです。

8.5.2 各オプションの機能

PIC/PID 機能と関連するオプションを説明します。
各オプションの機能詳細については、「コマンド・リファレンス」の章の各オプションの説明を参照ください。

(1) アプリケーションのコード生成 (pic,pid オプション)
pic オプションを有効にしてコンパイルすると、PIC 機能が有効になり、コード領域 (P セクション) が PIC になります。PIC は分岐先アドレスや関数アドレスの取得を全て PC 相対で行うため、リンク後も任意のアドレスに配置することができます。
pid オプションを有効にしてコンパイルすると、PID 機能が有効になり、ROM データ領域 (C_8, C, C_2, C_1, W, W_2, W_1 および L セクション) が PID になります。プログラムは PID に対しその先頭アドレスを示すレジスタ (PID レジスタ) から相対のアクセスでアクセスします。ユーザはマスタで PID レジスタの設定値を変化させて、リンク後も PID を任意のアドレスに移動することができます。
なお、PIC 機能 (pic オプション) と PID 機能 (pid オプション) は、それぞれ独立した機能として動作できるように設計しておりますが、同時に有効にしたうえで、PIC と PID を隣接させてご利用いただくことを推奨します。PIC 機能と PID 機能を個別に利用したり、PIC と PID の相対距離を変更したアプリケーションのデバッグは、デバッガのバージョンによりサポートされない場合があります。本書でも PIC 機能と PID 機能を同時に有効にした例で説明しています。

(2) 共有ライブラリ対応 (jump_entries_for_pic, nouse_pid_register オプション)
アプリケーションからマスタにあるライブラリを呼び出すための機能です。
nouse_pid_register オプションは、マスタのコンパイル時に使用し、PID レジスタを使用しないコードを生成します。
jump_entries_for_pic オプションを、マスタのリンク時に最適化リンケージエディタに指定すると、アプリケーションから固定アドレスにあるライブラリ関数を呼び出すためのジャンプテーブルを生成します。

(3) RAM 領域の共有 (Fsymbol オプション)
マスタ上の変数を、リンク単位の違うアプリケーションからでも読み書きできるようにするための機能です。
マスタのリンク時に、Fsymbol オプションを最適化リンケージエディタに指定すると、アプリケーションから固定アドレスで変数を参照するためのシンボルテーブルを生成します。

8.5.3 アプリケーションに関する制限事項

(1) RAM 領域
RAM 領域には PID 機能を適用できません。

(2) アプリケーションの同時実行
PIC/PID 機能を使うと、同じアプリケーションのコピーを複数個 ROM に置き、それぞれを実行できますが、RAM 領域が重なっているため、同じアプリケーションのコピーを同時に複数個実行することはできません。

(3) スタートアップ
アプリケーションに使用するスタートアップとしては、標準のスタートアップ（統合開発環境が生成。詳しくは「8.3 スタートアッププログラム」を参照）はそのままでは使用できません。「8.5.7 アプリケーションのスタートアップ」を参考に、スタートアップを作成してください。

8.5.4 PIC/PID 機能で必要なシステム依存処理

次の処理は、システム仕様に合わせてお客様にてご用意いただく必要があります。

(1) マスタの初期化
PIC/PID 機能を使用しない通常のプログラムと同様の処理を行います。

(2) マスタからアプリケーションを起動
アプリケーションの PID の先頭アドレスを PID レジスタに設定し、PIC の起動アドレスへ分岐することで、アプリケーションを起動します。

(3) アプリケーションの初期化
セクションの初期化を行い、アプリケーションの main 関数を実行します。

(4) アプリケーションの終了
main 関数が終了したら、マスタに処理を返します。

8.5.5 コード生成オプションの組み合わせ

マスタとアプリケーションをビルドするときは、構成するオブジェクト間で PIC/PID 機能に関係するオプション指定を合わせておく必要があります。
以下に、オブジェクトごとのコンパイル時のオプションの指定規則と、組み合わせて利用できるオブジェクトのオプション指定の制限について示します。

(1) マスタ
マスタをビルドするときは、PIC/PID 機能オプションを表 8.6 のように指定してください。

表 8.6 マスタ内の PIC/PID 機能オプションの指定規則

	オプション名	コンパイル時	リンク可能オブジェクトのオプション指定の条件
1	pic	× 指定不可	pic の指定なし
2	pid	× 指定不可	pid の指定なし
3	nouse_pid_register	△ 標準ライブラリ、スタートアップ内の PID レジスタ設定箇所以外は指定可	制限なし
4	fint_register	○ 指定可	同一パラメータの fint_register の指定が必須
5	base	○ 指定可	同一パラメータの base の指定が必須

(2) アプリケーション
アプリケーションをビルドするときは、PIC/PID 機能オプションを表 8.7 のように指定してください。

表 8.7 アプリケーション内の PIC/PID 機能オプションの指定規則

	オプション名	コンパイル時	リンク可能オブジェクトのオプション指定の条件
1	pic	○ 指定可	pic の指定が必須
2	pid	○ 指定可	pid の指定が必須
3	nouse_pid_register	× 指定不可	nouse_pid_register の指定なし
4	fint_register	○ 指定可	同一パラメータの fint_register の指定が必須
5	base	○ 指定可	同一パラメータの base*1 の指定が必須

注 1. pid 指定時は base=rom=< レジスタ > の指定はできません。

(3) マスタとアプリケーション間
マスタとアプリケーションはそれぞれ PIC/PID 機能オプションを表 8.8 のように指定する必要があります。

表 8.8 マスタとアプリケーション間の PIC/PID 機能オプションの組み合わせ規則

	アプリケーションのオプション	マスタのオプション
1	pic	制限なし
2	pid	アプリケーションからマスター上の関数を呼び出す場合は、nouse_pid_register が必須
3	fint_register	同一パラメータの fint_register が必須
4	base	同一パラメータの base*1 が必須

注 1. pid 指定時は base=rom=< レジスタ > の指定はできません。

8.5.6 マスタのスタートアップ

次の 2 点を除き、必要な処理は PIC/PID 機能を用いない通常のプログラムと同じです。「7.3 スタートアッププログラムの作成」に従ったスタートアップに、次の 2 つの内容を追加してください。

(1) アプリケーションの起動と復帰
main 関数で、PID レジスタを設定し、PIC のエントリアドレスに分岐してアプリケーションを起動します。また、アプリケーションからマスタに戻れる手段を用意しておく必要があります。

(2) 使用する共有ライブラリ関数の参照
アプリケーションが利用する共有ライブラリは、あらかじめマスタでも参照しておく必要があります。
以下に、main 関数から PIC/PID アプリケーションを呼び出す例を示します。
なお、この例は次の条件に基づきます。

- アプリケーションの終了時、RTS 命令でマスタに復帰できるものとします。
- アプリケーションは戻り値を持たないものとします。
- アプリケーションに対する PIC の起動アドレス (PIC_entry)、および PID の先頭アドレス (PID_address) は、マスタをビルドする時点で既知および固定であるとします。
- PID レジスタは R13 であるものとします。
- アプリケーション側のセクション領域の初期化は、マスタ側では行わないこととします。
- 共有ライブラリとして、アプリケーションは printf 関数のみ使用するものとします。

例

```
/* マスタ側プログラム */
/* PIC/PID アプリケーションを起動する */
/* ( アプリケーションが、戻り値を返さず、RTS で復帰するシステム仕様の場合 ) */
#include <stdio.h>
#pragma inline_asm Launch_PICPID_Application
void Launch_PICPID_Application(void *pic_entry, void *pid_address)
{
    MOV.L    R2,__PID_R13
    JSR      R1
}
int main()
{
    void *PIC_entry   = (void*)0x500000;  /* PIC の起動アドレス */
    void *PID_address = (void*)0x120000;  /* PID の先頭アドレス */

    /* (1) アプリケーションの起動と復帰 */
    Launch_PICPID_Application(PIC_entry, PID_address);

    return 0;
}

/* (2) アプリケーションで使用する共有ライブラリの参照 */
void *_dummy_ptr = (void*)printf;  /* printf 関数 */
```

8.5.7 アプリケーションのスタートアップ

アプリケーションでは、次の項目を設定してください。
【オプション】と書かれている項目は、不要な場合があります。

(1) エントリポイント (PIC の起動アドレス) の用意
アプリケーションの起動アドレスです。

(2) スタックポインタの初期化【オプション】
マスタとスタックを共有する場合は、不要です。
必要な場合は、7.3.2(2) を参考に、設定を追加してください。

(3) ベースレジスタに使用する汎用レジスタの初期化【オプション】
ベースレジスタを使用しない場合は、不要です。
必要な場合は、7.3.2(3) を参考に、設定を追加してください。

(4) セクションの初期化処理【オプション】
マスタ側で初期化する場合は、不要です。
必要な場合は、後述の例を参考に、設定を追加してください。
なお、7.3.2(5) の方法はそのままでは使用できません。

(5) ライブラリの初期化処理【オプション】
標準ライブラリを使用しない場合は、不要です。
必要な場合は、7.3.2(6) を参考に、設定を追加してください。

(6) main 関数向け PSW 初期化【オプション】
必要に応じて、割り込みマスクやユーザモードへの移行を行います。
7.3.2(8) と (9) を参考に、設定を追加してください。

(7) ユーザプログラムの実行
main 関数を実行します。
7.3.2(10) を参考に、設定してください。

以下、アプリケーション側のスタートアップ例を示します。
3 つのファイルに分かれています。

- startup_picpid.c … スタートアップ本体。
- initsct_pid.src … セクション初期化の PID 版である _INITSCT_PID です。
  7.3.2(5) で述べている _INITSCT 関数を PID 対応にしたものです。
  プログラム中の ”__PID_REG” は、PID レジスタに変換されます。
- initiolib.c … 標準ライブラリの初期化を行う、_INITLIB を収録しています。
  7.3.2(6) をアプリケーション用に変更したものです。

```
[startup_picpid.c]
// マニュアル 7.3.2(5) セクションの初期化処理
#pragma section C C$DSEC // セクション名を C$DSEC にします
const struct {
    void *rom_s; // 初期化データセクションの ROM 上の先頭アドレスメンバ
    void *rom_e; // 初期化データセクションの ROM 上の最終アドレスメンバ
    void *ram_s; // 初期化データセクションの RAM 上の先頭アドレスメンバ
} DTBL[] = {__sectop("D"), __secend("D"), __sectop("R")};
#pragma section C C$BSEC // セクション名を C$BSEC にします
const struct {
    void *b_s; // 未初期化データセクションの先頭アドレスメンバ
    void *b_e; // 未初期化データセクションの最終アドレスメンバ
} BTBL[] = {__sectop("B"), __secend("B")};

extern void main(void);
extern void _INITLIB(void); // マニュアル 7.3.2(6) ライブラリの初期化処理
#pragma entry application_pic_entry
void application_pic_entry(void)
{
    _INITSCT_PICPID();
    _INITLIB();
     main();
}

[initsct_pid.src]
; PID 対応 セクション初期化ルーチン
    .glb   __INITSCT_PICPID
    .glb   __PID_TOP
    .section  C$BSEC,ROMDATA,ALIGN=4
    .section  C$DSEC,ROMDATA,ALIGN=4
    .section  P,CODE

__INITSCT_PICPID:               ; function: _INITSCT
    .STACK    __INITSCT_PICPID=28
    PUSHM     R1-R6
    ADD       #-__PID_TOP,__PID_REG,R6   ;   How long distance PID moves
;;;
;;; clear BBS(B)
;;;
    ADD       #TOPOF C$BSEC,  R6, R4
    ADD       #SIZEOF C$BSEC, R4, R5
    MOV.L     #0, R2
    BRA       next_loop1
```

```
loop1:
    MOV.L     [R4+], R1
    MOV.L     [R4+], R3
    CMP       R1, R3
    BLEU      next_loop1
    SUB       R1, R3
    SSTR.B
next_loop1:
    CMP       R4,R5
    BGTU      loop1

;;;
;;; copy DATA from ROM(D) to RAM(R)
;;;
    ADD       #TOPOF C$DSEC,  R6, R4
    ADD       #SIZEOF C$DSEC, R4, R5
    BRA       next_loop3

loop3:
    MOV.L     [R4+], R2
    MOV.L     [R4+], R3
    MOV.L     [R4+], R1
    CMP       R2, R3
    BLEU      next_loop3
    SUB       R2, R3
    ADD       R6, R2       ; Adjust for real address of PID
    SMOVF
next_loop3:
    CMP       R4, R5
    BGTU      loop3
    POPM      R1-R6
    RTS

    .end
```

```
[initiolib.c]
#include <stdio.h>
#include <stdlib.h>
#define IOSTREAM 3
const size_t _sbrk_size = 520; // ヒープ領域確保サイズの最小単位を指定します
// ( 省略時：1024)

void _INIT_LOWLEVEL(void);
void _INIT_OTHERLIB(void);

void _INITLIB (void)
{
    _INIT_LOWLEVEL(); // 低水準インタフェースルーチンの初期設定をします
    _INIT_IOLIB(); // 入出力ライブラリの初期設定をします
    _INIT_OTHERLIB(); // rand 関数、strtok 関数の初期設定をします
}
void _INIT_LOWLEVEL(void)
{ // 低水準ライブラリで必要な初期設定をしてください
}
void _INIT_OTHERLIB(void)
{
     srand(1); // rand 関数を使用する場合の初期設定です
}
```

9. 関数呼び出し仕様

9.1 関数呼び出しインタフェース

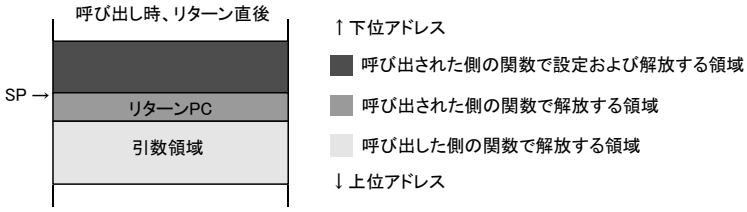
CC-RX の C/C++ 言語関数におけるプログラム呼び出し時の引数などの扱い方について説明します。
コンパイラのコード生成はこの内容に従って行っています。
アセンブラで C/C++ 言語とインタフェースをとる関数を作成する場合は、このルールに従う必要があります。
割り込み関数の場合は、4.2.4 拡張仕様の使用方法 の (3) 割り込み関数作成記述 も併せて参照ください。

9.1.1 スタックに関する規則

(1) スタックポインタ
スタックポインタの指すアドレスよりも下位 (0 番地の方向) のスタック領域に有効なデータを格納してはいけません。スタックポインタより下位アドレスに格納されたデータは、割り込み処理で破壊される可能性があります。

(2) スタックフレームの割り付け、解放
関数呼び出しが行われた時点 (JSR または BSR 命令の実行直後) では、スタックポインタは呼び出した関数側で使用したスタックの最下位アドレスを指しています。このスタックポインタの指している領域より上位アドレスのデータの割り付け、設定は呼び出す側の関数の役目です。
関数のリターン時は、呼び出された関数で確保した領域を解放してから、通常 RTS 命令を用いて呼び出した関数へ返ります。これより上位アドレスの領域 (リターン値アドレスおよび引数の領域) は、呼び出した側の関数で解放します。
図 9.1 は、関数呼び出し直後のスタックフレームの状態を説明したものです。

図 9.1 スタックフレームの割り付け、解放に関する規則



9.1.2 レジスタに関する規則

関数呼び出し前後において、レジスタの値が同一であることを保証するかどうかは、レジスタにより異なります。また、オプションにより特定の用途向けに使用するレジスタがあります。レジスタの使用規則を表 9.1 に示します。

表 9.1 レジスタ使用規則

レジスタ	関数呼び出し前後で値を保証	関数入口	関数出口	高速割り込み用レジスタ *1	ベースレジスタ *2	PIDレジスタ *3
R0	保証する	スタックポインタ	スタックポインタ	-	-	-
R1	保証しない	引数 1	戻り値 1	-	-	-
R2	保証しない	引数 2	戻り値 2	-	-	-
R3	保証しない	引数 3	戻り値 3	-	-	-
R4	保証しない	引数 4	戻り値 4	-	-	-
R5	保証しない	-	(不定)	-	-	-
R6	保証する	-	(入口の値を保持)	-	-	-
R7	保証する	-	(入口の値を保持)	-	-	-
R8	保証する	-	(入口の値を保持)	-	○	-
R9	保証する	-	(入口の値を保持)	-	○	○
R10	保証する	-	(入口の値を保持)	○	○	○
R11	保証する	-	(入口の値を保持)	○	○	○
R12	保証する	-	(入口の値を保持)	○	○	○
R13	保証する	-	(入口の値を保持)	○	○	○
R14	保証しない	-	(不定)	-	-	-
R15	保証しない	構造体戻り値へのポインタ	(不定)	-	-	-
DR0 ～ DR15 【V3.01.00 以降】	保証する	-	(入口の値を保持)	-	-	-
DCMR 【V3.01.00 以降】	保証する	-	(入口の値を保持)	-	-	-
DPSW DECNT DEPC 【V3.01.00 以降】	保証しない	-	(不定)	-	-	-
ISP USP	スタックポインタの場合は R0 と同じ。そうでない場合は変化しません。*4			-	-	-
PC	-	プログラムカウンタ *5		-	-	-
PSW	保証しない	-	(不定)	-	-	-
FPSW	保証しない	-	(不定)	-	-	-
ACC	保証しない *6	-	(不定) *6	-	-	-

レジスタ	関数呼び出し前後で値を保証	関数入口	関数出口	高速割り込み用レジスタ *1	ベースレジスタ *2	PIDレジスタ *3
ACC0 ACC1	保証しない *6	-	(不定) *6	-	-	-
INTB BPC BPSW FINTV	-	変化しません *4	-	-	-	-

注 1. R10 ～ R13 の 4 本は、fint_register オプションにより、一部または全部が「高速割り込み機能」に使われることがあります。「高速割り込み機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

注 2. R8 ～ R13 の 6 本は、base オプションにより、一部または全部が「ベースレジスタ機能」に使われることがあります。「ベースレジスタ機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

注 3. R9 ～ R13 のうちの 1 本は、pid オプションにより「PID 機能」に使われることがあります。「PID 機能」に割り当てられたレジスタは、他の用途に使用することはできません。機能の詳細はオプションの説明を参照してください。

注 4. 組み込み関数または #pragma inline_asm で、これらのレジスタを設定したり更新したりする場合を除きます。

注 5. 関数の呼び出しに使用する命令の仕様に従います。関数の呼び出しには、BSR, JSR, BRA および JMP のいずれかの命令を用います。

注 6. アキュムレータ (ACC, ACC0, ACC1) を更新する命令は、RX のソフトウェアマニュアルを参照してください。

9.1.3 引数の設定、参照に関する規則

引数に対する一般的な規則と、引数の割り付け方について述べます。
引数が実際どのように割り付けられるかは、「9.1.5 引数割り付けの具体例」を参照ください。

(1) 引数の渡し方
引数の値を、必ずレジスタまたはスタック上の引数の割り付け領域にコピーしたあとで関数を呼び出します。呼び出した側の関数では、リターン後に引数の割り付け領域を参照することはありませんので、呼び出された側の関数で引数の値を変更しても呼び出した側の処理は直接には影響を受けません。

(2) 型変換の規則

(a) 関数原型によって型が宣言されている引数は、宣言された型に変換します。

(b) 関数原型によって型が宣言されていない引数の型変換は、以下の規則に従って変換します。

- **2** バイト以下の整数型は、**4** バイト整数型に変換されます。
- **float** 型の引数は、**double** 型に変換します。
- 上記以外の引数は、変換しません。

例

```
void p(int,... );
void f( )
{
  char c;
  p(1.0, c);
}
```

cは、対応する引数の型宣言がないので、4バイト整数型に変換されます。

1.0は、対応する引数の型がint型なので、4バイト整数型に変換されます。

(3) 引数の割り付け領域
引数は、レジスタに割り付ける場合とスタック上の引数領域に割り付ける場合があります。引数の割り付け領域を図 9.2 に示します。

通常、ソースプログラムにおける引数の宣言順に、番号の小さいレジスタから順に割り付けを行い、レジスタが全て割り付いたらスタックに割り付けます。但し、可変個の引数を持つ関数など、レジスタが余っていてもスタックに割り付ける場合もあります。また、C++ プログラムの非静的関数メンバの this ポインタは、常に R1 に割り付けられます。
引数割り付け領域の一般規則を表 9.2 に示します。

図 9.2 スタックフレームの割り付け、解放に関する規則

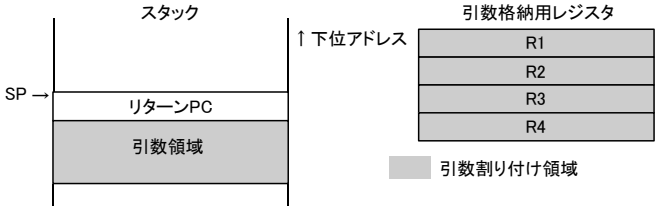


表 9.2 引数割り付け領域の一般規則

レジスタで渡される引数			スタック渡しになる引数
対象の型	引数格納用レジスタ	割り付け方	
signed char, (unsigned)char, bool, _Bool, (signed)short, unsigned short, (signed)int, unsigned int, (signed)long, unsigned long, float、double*1, long double*1, ポインタ , データメンバへのポインタ , リファレンス	R1～R4のうち1つ	signed char, (signed)short は符号拡張、(unsigned)char, unsigned short はゼロ拡張を行った結果を割り付ける その他の型はそのままレジスタに割り付ける	(1) 引数の型がレジスタ渡しの対象の型以外のもの (2) 関数原型により可変個の引数を持つ関数として宣言しているもの *3 (3) R1 ～ R4 のうち、まだ他の引数に割り当てられていないものの本数が、割り当てに必要な本数より少ない場合
(signed)long long, unsigned long long, double*2, long double*2	R1～R4のうち2つ	下位 4 バイトを番号の少ない方に、上位 4 バイトを番号の大きい方に割り付ける	
16バイト以内でサイズが4の倍数の構造体型、共用体型、クラス型	R1 ～ R4 のうち、サイズを 4 で割った数	メモリイメージの先頭から 4 バイトずつ、レジスタ番号が増える方向に割り付ける	

注 1. dbl_size=8 を指定しなかった場合です。

注 2. dbl_size=8 を指定した場合です。

注 3. 関数原型により可変個の引数をもつ関数として宣言している場合、宣言の中で対応する型のない引数およびその直前の引数はスタック渡しになります。型のない引数は、2 バイト以下の整数は long 型に、float 型は double 型にそれぞれ変換して、全て境界調整数が 4 の引数として取り扱います。

例

```
int f2(int,int,int,int,...);
:
f2(a,b,c,x,y,z); → x、y、z はスタック渡しになります。
```

(4) スタック渡しとなる引数の割り付け方
表 9.2 で、スタック渡しとなる引数の、配置アドレス、およびスタックへの配置の仕方は以下となります。

- 各引数は、その境界調整数に応じたアドレスに配置します。
- 引数並びの左から右の順に、スタックが深くなる方向に配置されるように、スタックの引数用領域に格納します。すなわち、引数 A とその右隣の引数 B がともにスタック渡しとなる場合、引数 B のアドレスは、引数 A の配置アドレスに引数 A の占有サイズを加えたアドレスを、引数 B の境界調整数に整合させたアドレス、となります。

9.1.4 リターン値の設定、参照に関する規則

リターン値に対する一般的な規則と、リターン値の設定場所について述べます。

(1) リターン値の型変換
リターン値は、その関数の返す型に変換します。

例

```
long f();
long f()
{
    float x;
    return x; // 関数原型にしたがってリターン値は long 型に変換されます。
}
```

(2) リターン値の設定場所
関数のリターン値の型によっては、リターン値をレジスタに設定する場合とメモリに設定する場合があります。
リターン値の型と設定場所の関係は表 9.3 を参照してください。

表 9.3 リターン値の型と設定場所

No.	リターン値の型	リターン値の設定場所
1	singed char, (unsigned)char, (singed)short, unsigned short, (singed)int, unsigned int, (signed)long, unsigned long, float, double*2, long double*2, ポインタ, bool, _Bool, リファレンス , データメンバへのポインタ	R1 但し、signed char, (signed)short は符号拡張、(unsigned)char, unsigned shortはゼロ拡張を行った結果を設定
2	double*3, long double*3, (signed)long long, unsigned long long	R1, R2 下位 4 バイトを R1 に、上位 4 バイトを R2 に設定
3	16 バイト以内かつ 4 の倍数であるサイズの構造体、共用体、クラス型	メモリイメージの先頭から 4 バイトずつ R1,R2,R3,R4 の順に設定
4	3. 以外の構造体、共用体、クラス型	リターン値設定領域 (メモリ)*1

注 1. 関数のリターン値をメモリに設定する場合、リターン値はリターン値アドレスの指す領域に設定します。呼び出す側では、引数領域のほかにリターン値設定領域を確保し、そのアドレスを R15 に設定してから関数を呼び出します。

注 2. dbl_size=8 を指定しなかった場合です。

注 3. dbl_size=8 を指定した場合です。

9.1.5 引数割り付けの具体例

引数割り付けの具体例を示します。なお、アドレスは全ての図で右から左に向かって増加します (左側が上位アドレス)。

例 1. レジスタ渡しの対象の型である引数は、宣言順にレジスタ R1 ～ R4 に割り付けます。
途中でレジスタ渡しとならない引数があった場合、それ以降の引数はレジスタ渡しの対象となります。
スタック上では、その引数の境界調整数に補正したアドレスに配置されます。

```
int f(
    unsigned char,
    long long,
    long long,
    short,
    int,
    char,
    short,
    char,
    char,
    short);
        :
f(1,2,3,4,5,6,7,8,9,10);
 /*
** 1, 2, 4 がレジスタ渡しとなる
*/
```

<レジスタ>

R1	0x000000(ゼロ拡張)	0x01
R2	0x00000002	
R3	0x00000000	
R4	0x0000(符号拡張)	0x0004

<スタック>

*(R0+0)	0x00000003		
*(R0+4)	0x00000000		
*(R0+8)	0x00000005		
*(R0+12)	0x0007	空領域	0x06
*(R0+16)	0x000A	0x09	0x08

例 2. サイズが 16 バイト以下、かつ 4 の倍数である構造体および共用体型の引数は、レジスタ渡しの対象となります。それ以外の構造体および共用体型の引数は、スタック渡しとなります。

```
union  U { int a[2]; int b; } u;
struct S { short d; char c[4]; } s;
struct T { char g;  char f[2]; char e; } t;
int f(union U, struct S, struct T);
...
f(u, s, t);

 /*
 ** uは8バイトなのでレジスタ渡し
 ** sは6バイトなのでスタック渡し
 ** tは4バイトなのでレジスタ渡し
 */
```

<レジスタ>

R1	u.a[0] (=u.b)			
R2	u.a[1]			
R3	t.e	t.f[1]	t.f[0]	t.g

<スタック>

*(R0+0)	s.c[1]	s.c[0]	s.d	
*(R0+4)	空領域		s.c[3]	s.c[2]

例 3. 関数原型により可変個の引数を持つ関数として宣言している場合、対応する型のない引数およびその直前の引数は、宣言順にスタック渡しになります。

```
int f(int, float, int, int, ...)
    :
f(0, 1.0, 2, 3, 4)
```

<レジスタ>

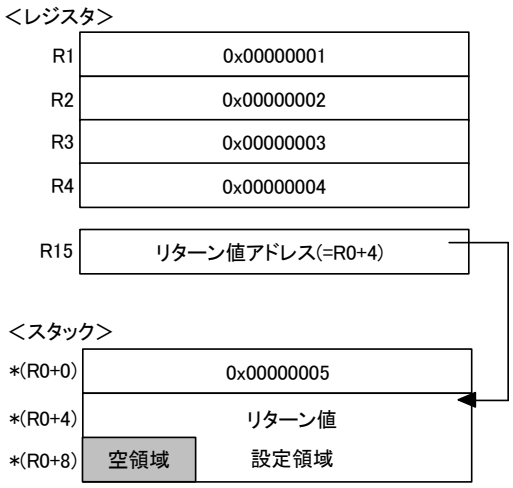
R1	0x00000000
R2	0x3F800000
R3	0x00000002

<スタック>

*(R0+0)	0x00000003
*(R0+4)	0x00000004

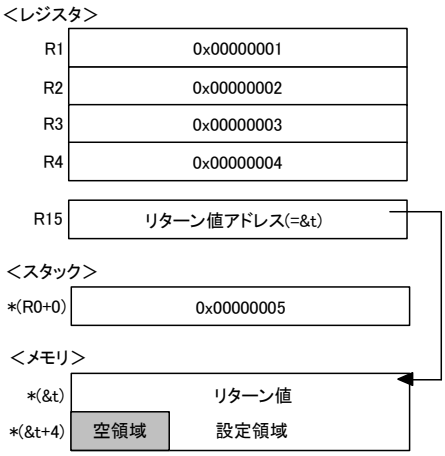
例 4. 関数の返す型が 16 バイトを超える、または 4 の倍数でないサイズの構造体または共用体型の場合、R15 にリターン値アドレスを設定します。

```
struct S{char a[7];};
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
f(1,2,3,4,5);
```



例 5. リターン値をメモリに設定する場合、通常例 4 のようにスタックを確保して設定しますが、リターン値を変数に設定する場合、スタックを確保せず、その変数のメモリ領域に直接設定します。この場合、R15 にはその変数のアドレスを設定します。

```
struct S{char a[7];}t;
struct S f(
   int a1,
   int a2,
   int a3,
   int a4,
   int a5);
   :
t=f(1,2,3,4,5);
```



9.2 コンパイラとアセンブラの外部名の相互参照方法

C/C++ プログラムの中で外部名として宣言されたものは、アセンブリプログラムとの間で相互に参照あるいは更新することができます。コンパイラは、次のものを外部名として扱います。

- 大域変数であって、かつ static 記憶クラスでないもの (C/C++ プログラム)
- extern 記憶クラスで宣言されている変数名 (C/C++ プログラム)
- static 記憶クラスを指定されていない関数名 (C プログラム)
- static 記憶クラスを指定されてない非メンバ非インライン関数名 (C++ プログラム)
- 非インラインメンバ関数名 (C++ プログラム)
- 静的データメンバ名 (C++ プログラム)

9.2.1 アセンブリプログラムの外部名を C/C++ プログラムで参照

アセンブリプログラムでは、.GLB を用いてシンボル名 (先頭に下線 "_" を付与) を外部定義宣言します。
C/C++ プログラムでは、シンボル名 (先頭に下線 "_" がない) を「extern」宣言します。
例 アセンブリソース

```
            .glb _a, _b
            .SECTION D,ROMDATA,ALIGN=4
_a:         .LWORD 1
_b:         .LWORD 1
            .END
```

例　C ソース

```
extern int a,b;
void f()
{
        a+=b;
}
```

9.2.2 C/C++ プログラムの外部 (変数および C 関数) 名をアセンブリプログラムで参照

C/C++ プログラムでは、変数名 (先頭に下線 "_" がない) を外部定義します。
アセンブリプログラムでは、.GLB を用いて外部名 (先頭に下線 "_" を付与) を外部参照宣言します。
例　C ソース

```
int a;
```

例　アセンブリソース

```
        .GLB _a
        .SECTION P,CODE
        MOV.L #A_a,R1
        MOV.L [R1],R2
        ADD #1,R2
        MOV.L R2,[R1]
        RTS
        .SECTION D,ROMDATA,ALIGN=4
A_a:    .LWORD _a
        .END
```

9.2.3 C++ プログラムの外部 (関数) 名をアセンブリプログラムで参照

アセンブリプログラムで参照する関数を「extern "C"」を用いて宣言することにより、(2) と同じ規則で参照できます。ただし、「extern "C"」を用いて宣言した関数は多重定義できません。
例　C ソース

```
extern "C"
void sub()
{
        :
}
```

例　アセンブリソース

```
.GLB _sub
.SECTION P,CODE
  :
PUSH.L R13
MOV.L 4[R0],R1
MOV.L R3,R12
MOV.L #_sub,R14
JSR R14
POP R13
RTS
  :
.END
```

10. メッセージ

10.1 概　　説

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、MISRA-C 検出メッセージについて説明します。

10.2 出力形式

(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 コンポーネント番号 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 コンポーネント番号 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
コンポーネント番号）:05
メッセージ番号　　:　5 桁の数値

10.3 メッセージ種別

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（CC-RX（V2.00.00 以上））

メッセージ種別	説明
C	内部エラー：処理を中止します。 出力オブジェクトは生成しません。
E	エラー：一定数以上発生した場合、処理を中止します。 出力オブジェクトは生成しません。
F	致命的エラー：処理を中止します。 出力オブジェクトは生成しません。
M	インフォメーション：情報を通知します。 メッセージを確認してください。その後、処理を継続します。
W	ワーニング：処理を続行します。 出力オブジェクトを生成します（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

メッセージ番号は、コンポーネント番号（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

この章では、ルネサスツール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

C0510000	［メッセージ］	Internal error.
	［対処方法］	特約店、または当社までご連絡ください。
C0511200	［メッセージ］	Internal error(*error-information*).
	［説明］	内部エラーが発生しました（*内容*）。
C0519996	［メッセージ］	Out of memory.
	［説明］	ccrx 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］	ccrx 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。
C0554098	［メッセージ］	Internal error
	［説明］	アセンブラ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。

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］	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。
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］	"*path*" specified by the "*character string*" option is a folder. Specify an input file.
	［説明］	" *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］	The file "*file*" specified by the "*character string*" option is not found
	［説明］	" *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］	"*path*" specified by the "*character string*" option is a folder. Specify an output file.
	［説明］	" *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］	The output folder "*folder*" specified by the "*character string*" option is not found.
	［説明］	" *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］	"*path*" specified by the "*character string*" option is not found.
	［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］	The "*character string*" option is not recognized.
	［説明］	" *文字列* " は認識されないオプションです。
E0511109	［メッセージ］	The "*character string*" option can not have an argument.
	［説明］	" *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］	The "*character string*" option requires an argument.
	［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。
E0511111	［メッセージ］	The "*character string*" option can not have a parameter.
	［説明］	" *文字列* " オプションにパラメータは指定できません。
E0511112	［メッセージ］	The "*character string*" option requires a parameter.
	［説明］	" *文字列* " オプションはパラメータが必要です。パラメータを指定してください。
E0511113	［メッセージ］	Invalid argument for the "*character string*" option.
	［説明］	" *文字列* " オプションに指定された引数が不正です。
E0511117	［メッセージ］	Invalid parameter for the "*character string*" option.
E0511118	［メッセージ］	Symbol is required for the "*character string*" option.
	［説明］	" *文字列* " オプションにはシンボルを指定してください。
E0511120	［メッセージ］	Specify a value (*value1* - *value2*) for the "*character string*" option.
	［説明］	指定したオプションのパラメータが指定可能範囲外です。
	［対処方法］	サイズ・オプションの値は、最小値から最大値の間で指定してください。
E0511122	［メッセージ］	The argument for the "*character string*" option must be an object file.
	［説明］	" *文字列* " オプションにはオブジェクト・ファイルを指定してください。
E0511127	［メッセージ］	The specified device is not supported.
	［説明］	サポートしていないデバイスが指定されました。

E0511129	［メッセージ］	Command file "*file*" is read more than once.
	［説明］	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］	Command file "*file*" cannot be read.
	［説明］	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］	Syntax error in command file "*file*".
	［説明］	コマンド・ファイル " *ファイル名* " の構文が認識できません。
E0511132	［メッセージ］	Failed to create temporary folder.
	［説明］	テンポラリ・フォルダを作成できません。
E0511133	［メッセージ］	The parameter for the "*character string*" option must be a folder when multiple source files are specified.
	［説明］	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］	Input file "*file*" is not found.
	［説明］	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］	"*path*" specified as an input file is a folder.
	［説明］	指定された入力ファイル " *パス名* " はフォルダです。
E0511136	［メッセージ］	Failed to delete a temporary file "*file*".
	［説明］	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0511137	［メッセージ］	Failed to delete a temporary folder "*folder*".
	［説明］	テンポラリ・フォルダ " *フォルダ名* " の削除に失敗しました。
E0511138	［メッセージ］	Failed to open an input file "*file*".
	［説明］	入力ファイル " *ファイル名* " のオープンに失敗しました。
E0511139	［メッセージ］	Failed to open an output file "*file*".
	［説明］	出力ファイル " *ファイル名* " のオープンに失敗しました。
E0511140	［メッセージ］	Failed to close an input file "*file*".
	［説明］	入力ファイル " *ファイル名* " のクローズに失敗しました。
E0511141	［メッセージ］	Failed to write an output file "*file*".
	［説明］	出力ファイル " *ファイル名* " の書き込みに失敗しました。
E0511142	［メッセージ］	Multiple source files are not allowed when the "*character string*" option is specified.
	［説明］	" *文字列* " オプションには入力ファイルを複数指定できません。
E0511145	［メッセージ］	"*character string2*" specified in the "*character string1*" option is not available.
	［説明］	" *文字列 1* " オプションで指定された " *文字列 2* " は使用できません。
E0511148	［メッセージ］	"*file name*" is specified as an output file for the different options.
	［説明］	ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。
	［対処方法］	ファイル " *ファイル名* " が複数の出力先指定オプションで同時に指定されています。異なるファイル名を指定してください。

E0511150	［メッセージ］	The "*character string1*" option and the "*character string2*" option are inconsistent.
	［説明］	" *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］	The "*character string1*" option needs the "*character string2*" option.
	［説明］	" *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］	Component file "*file name*" for the CC-RX is not found. Reinstall the CC-RX.
	［説明］	CC-RX 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。
E0511155	［メッセージ］	The "*character string*" option needs other option(s).
	［説明］	" *文字列* " オプションには他のオプションが必要です。
E0511157	［メッセージ］	The "*character string1*" option or the "*character string2*" option must be specified for this device.
	［説明］	このデバイスには、" *文字列 1*" オプションもしくは、" *文字列 2*" オプションが必要です。
E0511158	［メッセージ］	The "*character string*" option is not supported for this device.
	［説明］	このデバイスは、" *文字列* " オプションをサポートしていません。
E0511159	［メッセージ］	When the "*character string*" option is specified, source files cannot be input.
	［説明］	" *文字列* " オプションが指定されている時はソース・ファイルは指定できません。
E0511160	［メッセージ］	The "*character string*" option must be specified for this device.
	［説明］	このデバイスには、" *文字列* " オプションが必要です。
E0511161	［メッセージ］	Failed to delete a file "*file*".
	［説明］	ファイル " *ファイル名* " の削除に失敗しました。
E0511167	［メッセージ］	Illegal section naming.
	［説明］	セクションの命名に誤りがあります。用途の異なるセクションに同じ名前を付けています。
E0511173	［メッセージ］	Failed to access a temporary file
	［説明］	テンポラリ・ファイルの読み書きに失敗しました。
E0511175	［メッセージ］	Neither isa nor cpu is specified.
E0511176	［メッセージ］	Both "-isa" option and "-cpu" option are specified.
E0511178	［メッセージ］	" *文字列* " option is unavailable because the license of *バージョン* Professional edition is not found. Please consider purchasing the product of Professional edition.
	［説明］	*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。
E0511200	［メッセージ］	Internal error(*error-information*).
	［説明］	内部エラーが発生しました (*内容*)。
E0512001	［メッセージ］	Failed to delete a temporary file "*file-name*".
	［説明］	テンポラリ・ファイル " *ファイル名* " の削除に失敗しました。
E0520001	［メッセージ］	Last line of file ends without a newline.
	［対処方法］	ファイルの最終行が改行で終了していません。改行を追加してください。

E0520002	［メッセージ］	Last line of file ends with a backslash.
	［対処方法］	ファイルの最終行の最後にバックスラッシュがあります。削除してください。
E0520005	［メッセージ］	Could not open source file "*file*".
	［説明］	ソース・ファイル " *ファイル名* " を開くことができません。
E0520006	［メッセージ］	Comment unclosed at end of file.
	［説明］	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］	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］	Unrecognized token.
	［説明］	不明なトークンがあります。
	［対処方法］	該当箇所を確認してください。
E0520008	［メッセージ］	Missing closing quote.
	［説明］	文字列のクォーテーションが閉じられていません。
	［対処方法］	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］	# not expected here.
	［説明］	"#" はここには書けません。"#" が正しくない位置に記述されています。
E0520011	［メッセージ］	Unrecognized preprocessing directive.
	［説明］	不明な前処理指令があります。
E0520012	［メッセージ］	Parsing restarts here after previous syntax error
	［説明］	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］	Expected a file name
	［説明］	ファイル名がありません。
E0520014	［メッセージ］	Extra text after expected end of preprocessing directive.
	［説明］	前処理指令の後に不正な文字があります。
E0520017	［メッセージ］	Expected a "]".
	［説明］	"]" がありません。
E0520018	［メッセージ］	Expected a ")".
	［説明］	")" がありません。
E0520019	［メッセージ］	Extra text after expected end of number.
	［説明］	数値の後に不正な文字があります。
E0520020	［メッセージ］	Identifier "*character string*" is undefined.
	［説明］	識別子 " *文字列* " は定義されていません
E0520022	［メッセージ］	Invalid hexadecimal number.
	［説明］	不正な 16 進数です。
E0520023	［メッセージ］	Integer constant is too large.
	［説明］	定数の値が大きすぎます。

E0520024	[メッセージ]	Invalid octal digit.
	[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]	Quoted string should contain at least one character.
	[説明]	引用文字列は少なくとも 1 文字を含まなければなりません。
E0520026	[メッセージ]	Too many characters in character constant.
	[説明]	文字定数中の文字が多すぎます。
E0520027	[メッセージ]	Character value is out of range.
	[説明]	char 型の値が範囲を越えています。
E0520028	[メッセージ]	Expression must have a constant value.
	[説明]	式は定数値を持つ必要があります。
E0520029	[メッセージ]	Expected an expression.
	[説明]	式がありません。
E0520030	[メッセージ]	Floating constant is out of range.
	[説明]	浮動小数点数定数値が範囲を越えています。
E0520031	[メッセージ]	Expression must have integral type.
	[説明]	式は整数型を持つ必要があります。
E0520032	[メッセージ]	Expression must have arithmetic type.
	[説明]	式は算術型を持つ必要があります。
E0520033	[メッセージ]	Expected a line number.
	[説明]	#line の後の行番号がありません。
E0520034	[メッセージ]	Invalid line number.
	[説明]	#line の後の行番号が不正です。
E0520036	[メッセージ]	The #if for this directive is missing.
	[説明]	この前処理指令のための #if がありません。
E0520037	[メッセージ]	The #endif for this directive is missing.
	[説明]	この前処理指令のための #endif がありません。
E0520038	[メッセージ]	Directive is not allowed -- an #else has already appeared.
	[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]	Division by zero.
	[説明]	0 で除算を行いました。
E0520040	[メッセージ]	Expected an identifier.
	[説明]	識別子がありません。
E0520041	[メッセージ]	Expression must have arithmetic or pointer type.
	[説明]	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]	Operand types are incompatible (" *型 1*" and " *型 2*").
	[説明]	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]	Expression must have pointer type.
	[説明]	式はポインタ型を持つ必要があります。
E0520045	[メッセージ]	#undef may not be used on this predefined name.
	[説明]	既定義名に対して #undef を使用できません。
E0520046	[メッセージ]	symbol is predefined; attempted redefinition ignored
	[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]	Incompatible redefinition of macro "*macro*".
	[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]	Duplicate macro parameter name.
	[説明]	マクロの引数名が重複しています。
E0520050	[メッセージ]	## may not be first in a macro definition.
	[説明]	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]	## may not be last in a macro definition.
	[説明]	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]	Expected a macro parameter name.
	[説明]	マクロの引数名がありません。
E0520053	[メッセージ]	Expected a ":".
	[説明]	":" がありません。
E0520054	[メッセージ]	Too few arguments in macro invocation.
	[説明]	マクロに対する引数が足りません。
E0520055	[メッセージ]	Too many arguments in macro invocation.
	[説明]	マクロに対する引数が多すぎます。
E0520056	[メッセージ]	Operand of sizeof may not be a function.
	[説明]	sizeof のオペランドに関数は書けません。
E0520057	[メッセージ]	This operator is not allowed in a constant expression.
	[説明]	この演算子は定数式では使用できません。
E0520058	[メッセージ]	This operator is not allowed in a preprocessing expression.
	[説明]	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]	Function call is not allowed in a constant expression.
	[説明]	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]	This operator is not allowed in an integral constant expression.
	[説明]	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]	Integer operation result is out of range.
	[説明]	整数演算の結果が範囲を越えました。
E0520062	[メッセージ]	Shift count is negative.
	[説明]	シフト数が負数です。

E0520063	[メッセージ]	Shift count is too large.
	[説明]	シフト数が多すぎます。
E0520064	[メッセージ]	Declaration does not declare anything.
	[説明]	宣言は何も宣言できません。
E0520065	[メッセージ]	Expected a ";".
	[説明]	";" がありません。
E0520066	[メッセージ]	Enumeration value is out of "int" range.
	[説明]	enum の値が "int" の範囲を越えています。
E0520067	[メッセージ]	Expected a "}".
	[説明]	"}" がありません。
E0520069	[メッセージ]	Integer conversion resulted in truncation
	[説明]	整数型の変換において、切り捨てが生じます。
E0520070	[メッセージ]	Incomplete type is not allowed.
	[説明]	不完全型は許されていません。
E0520071	[メッセージ]	Operand of sizeof may not be a bit field.
	[説明]	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]	Operand of "*" must be a pointer.
	[説明]	"*" 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]	This declaration has no storage class or type specifier.
	[説明]	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]	A parameter declaration may not have an initializer.
	[説明]	引数宣言に初期化子は書けません。
E0520079	[メッセージ]	Expected a type specifier.
	[説明]	型指定子がありません。
E0520080	[メッセージ]	A storage class may not be specified here.
	[説明]	記憶域クラスはここでは指定できません。
E0520081	[メッセージ]	More than one storage class may not be specified.
	[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]	Type qualifier specified more than once.
	[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]	Invalid combination of type specifiers.
	[説明]	不正な型指定子の組み合わせです。
E0520085	[メッセージ]	Invalid storage class for a parameter.
	[説明]	引数に対する記憶域クラスが不正です。
E0520086	[メッセージ]	Invalid storage class for a function.
	[説明]	関数に対する記憶域クラスが不正です。

E0520087	［メッセージ］	A type specifier may not be used here.
	［説明］	型指定子はここでは使用できません。
E0520088	［メッセージ］	Array of functions is not allowed.
	［説明］	関数の配列は許されていません。
E0520089	［メッセージ］	Array of void is not allowed.
	［説明］	void 型の配列は許されていません。
E0520090	［メッセージ］	Function returning function is not allowed.
	［説明］	関数を返す関数は許されていません。
E0520091	［メッセージ］	Function returning array is not allowed.
	［説明］	配列を返す関数は許されていません。
E0520092	［メッセージ］	Identifier-list parameters may only be used in a function definition.
	［説明］	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］	Function type may not come from a typedef.
	［説明］	関数型は typedef に指定できません。
E0520094	［メッセージ］	The size of an array must be greater than zero.
	［説明］	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］	Array is too large.
	［説明］	配列が大きすぎます。
E0520096	［メッセージ］	A translation unit must contain at least one declaration.
	［説明］	コンパイル単位は少なくとも 1 つの宣言を含まなければなりません。
E0520097	［メッセージ］	A function may not return a value of this type.
	［説明］	この型の値は関数返却値にできません。
E0520098	［メッセージ］	An array may not have elements of this type.
	［説明］	この型の配列は許されていません。
E0520099	［メッセージ］	A declaration here must declare a parameter.
	［説明］	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］	Duplicate parameter name.
	［説明］	引数名が重複しています。
E0520101	［メッセージ］	" *シンボル名* " has already been declared in the current scope.
	［説明］	" *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］	Forward declaration of enum type is nonstandard.
	［説明］	列挙型の前方宣言は標準でありません。
E0520103	［メッセージ］	Class is too large.
	［説明］	クラスが大きすぎます。
E0520104	［メッセージ］	Struct or union is too large.
	［説明］	構造体または共用体が大きすぎます。

E0520105	［メッセージ］	Invalid size for bit field.
	［説明］	ビット・フィールドのサイズが不正です。
E0520106	［メッセージ］	Invalid type for a bit field.
	［説明］	ビット・フィールドの型が不正です。
E0520107	［メッセージ］	Zero-length bit field must be unnamed.
	［説明］	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］	Expression must have (pointer-to-) function type.
	［説明］	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］	Expected either a definition or a tag name.
	［説明］	タグ名または定義がありません。
E0520112	［メッセージ］	Expected "while".
	［説明］	"while" がありません。
E0520114	［メッセージ］	*Type* "*symbol*" was referenced but not defined.
	［説明］	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］	A continue statement may only be used within a loop.
	［説明］	continue 文はループの中でのみ使用できます。
E0520116	［メッセージ］	A break statement may only be used within a loop or switch.
	［説明］	break 文はループまたは switch の中でのみ使用できます。
E0520118	［メッセージ］	A void function may not return a value.
	［説明］	void 関数は値を返せません。
E0520119	［メッセージ］	Cast to type "*type*" is not allowed.
	［説明］	"*type*" 型へのキャストは許されていません。
E0520120	［メッセージ］	Return value type does not match the function type.
	［説明］	返却値の型が関数の型と合っていません。
E0520121	［メッセージ］	A case label may only be used within a switch.
	［説明］	case ラベルは switch の中でのみ使用できます。
E0520122	［メッセージ］	A default label may only be used within a switch.
	［説明］	default ラベルは switch の中でのみ使用できます。
E0520123	［メッセージ］	case label value has already appeared in this switch.
	［説明］	この case ラベルの値はすでにこの switch の中で使用されています。
E0520124	［メッセージ］	default label has already appeared in this switch.
	［説明］	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］	Expected a "(".
	［説明］	"(" がありません。
E0520126	［メッセージ］	Expression must be an lvalue.
	［説明］	式は左辺値である必要があります。

E0520127	［メッセージ］	Expected a statement.
	［説明］	文がありません。
E0520129	［メッセージ］	A block-scope function may only have extern storage class.
	［説明］	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］	Expected a "{".
	［説明］	"{" がありません。
E0520131	［メッセージ］	Expression must have pointer-to-class type.
	［説明］	式はクラスへのポインタ型でなければなりません。
E0520132	［メッセージ］	Expression must have pointer-to-struct-or-union type.
	［説明］	式は構造体か共用体へのポインタでなければなりません。
E0520133	［メッセージ］	Expected a member name.
	［説明］	メンバ名が不正です。または、ありません。
E0520134	［メッセージ］	Expected a field name.
	［説明］	フィールド名がありません。
E0520135	［メッセージ］	*symbol* has no member *member*.
	［説明］	*symbol* はメンバ *member* を持ちません。
E0520136	［メッセージ］	*種別* " *シンボル名* " has no field " *フィールド名* ".
	［説明］	*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］	Expression must be a modifiable lvalue.
	［説明］	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］	Taking the address of a register variable is not allowed.
	［説明］	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］	Taking the address of a bit field is not allowed.
	［説明］	ビット・フィールドに対するアドレス演算子は許されていません。
E0520140	［メッセージ］	Too many arguments in function call.
	［説明］	関数呼び出しに対する引数が多すぎます。
E0520141	［メッセージ］	Unnamed prototyped parameters not allowed when body is present.
	［説明］	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］	Expression must have pointer-to-object type.
	［説明］	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］	A value of type " *型名 1*" cannot be used to initialize an entity of type " *型名 2*".
	［説明］	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］	*Type* "*symbol*" may not be initialized.
	［説明］	*Type* "*symbol*" は初期化できません。
E0520146	［メッセージ］	Too many initializer values.
	［説明］	初期化子が多すぎます。

E0520147	[メッセージ]	Declaration is incompatible with "*declaration*".
	[説明]	宣言は "*declaration*" と整合しません。
E0520148	[メッセージ]	*Tyep* "*symbol*" has already been initialized.
	[説明]	*Tyep* "*symbol*" はすでに初期化されています。
E0520149	[メッセージ]	A global-scope declaration may not have this storage class.
	[説明]	グローバル・スコープの宣言ではこの記憶域クラスを指定できません。
E0520150	[メッセージ]	A type name may not be redeclared as a parameter.
	[説明]	型名は引数として再宣言できません。
E0520151	[メッセージ]	A *typedef name* may not be redeclared as a parameter.
	[説明]	*typedef name* は引数として再宣言できません。
E0520153	[メッセージ]	Expression must have class type.
	[説明]	式はクラス型である必要があります。
E0520154	[メッセージ]	Expression must have struct or union type.
	[説明]	式は構造体または共用体型である必要があります。
E0520157	[メッセージ]	Expression must be an integral constant expression.
	[説明]	式は整数定数式である必要があります。
E0520158	[メッセージ]	Expression must be an lvalue or a function designator.
	[説明]	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]	Declaration is incompatible with previous "*symbol*".
	[説明]	宣言は以前の "*symbol*" と整合しません。
E0520160	[メッセージ]	External name conflicts with external name of "*symbol*".
	[説明]	外部名が "*symbol*" と矛盾しています。
E0520165	[メッセージ]	Too few arguments in function call.
	[説明]	関数呼び出しに引数が足りません。
E0520166	[メッセージ]	Invalid floating constant.
	[説明]	不正な浮動小数点定数です。
E0520167	[メッセージ]	Argument of type "*type1*" is incompatible with parameter of type "*type2*".
	[説明]	"*type1*" 型の引数は型 "*type2*" の引数と整合しません。
E0520168	[メッセージ]	A function type is not allowed here.
	[説明]	関数型はここでは許されていません。
E0520169	[メッセージ]	Expected a declaration.
	[説明]	宣言がありません。
E0520170	[メッセージ]	Pointer points outside of underlying object.
E0520171	[メッセージ]	Invalid type conversion.
	[説明]	不正な型変換です。
E0520172	[メッセージ]	External/internal linkage conflict with previous declaration.

E0520173	［メッセージ］	Floating-point value does not fit in required integral type.
	［説明］	浮動小数点数は要求された整数型に入りません。
E0520179	［メッセージ］	Right operand of "%" is zero.
	［説明］	"%" の右オペランドが 0 です。
E0520183	［メッセージ］	Type of cast must be integral.
	［説明］	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］	Type of cast must be arithmetic or pointer.
	［説明］	キャストの型は算術型かポインタ型である必要があります。
E0520194	［メッセージ］	Expected an asm string.
	［説明］	__asm() 中にアセンブラ文字列がありません。
E0520195	［メッセージ］	An asm function must be prototyped.
	［説明］	asm 関数はプロトタイプ宣言される必要があります。
E0520196	［メッセージ］	An asm function may not have an ellipsis.
	［説明］	asm 関数に省略記号は使用できません。
E0520220	［メッセージ］	Integral value does not fit in required floating-point type.
	［説明］	整数値が要求された浮動小数点型に入りません。
E0520221	［メッセージ］	Floating-point value does not fit in required floating-point type.
	［説明］	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］	Floating-point operation result is out of range.
	［説明］	浮動小数点演算の結果が範囲を越えました。
E0520227	［メッセージ］	Macro recursion.
	［説明］	マクロが再帰しています。
E0520228	［メッセージ］	Trailing comma is nonstandard.
	［説明］	最後のカンマは標準ではありません
E0520230	［メッセージ］	Nonstandard type for a bit field.
	［説明］	ビット・フィールドに対し標準でない型です。
E0520235	［メッセージ］	Variable *変数名* was declared with a never-completed type.
	［説明］	変数 " *変数名* " が不完全型で宣言されました。
E0520238	［メッセージ］	Invalid specifier on a parameter
	［説明］	引数の指定子が不正です。
E0520239	［メッセージ］	Invalid specifier outside a class declaration.
	［説明］	クラス宣言外で不正な指定子を使用しています。
E0520240	［メッセージ］	Duplicate specifier in declaration.
	［説明］	宣言の指示子が重複しています。
E0520241	［メッセージ］	A union is not allowed to have a base class.
	［説明］	union 型は基底クラスを持つことはできません。

E0520242	［メッセージ］	Multiple access control specifiers are not allowed.
	［説明］	アクセス指定子が重複して使われています。
E0520243	［メッセージ］	class or struct definition is missing.
	［説明］	class 定義の括弧の対応がとれません。
E0520244	［メッセージ］	Qualified name is not a member of class *型* or its base classes.
	［説明］	限定名がクラスまたは基底クラスのメンバの " *型* " でありません。
E0520245	［メッセージ］	A nonstatic member reference must be relative to a specific object.
	［説明］	非静的メンバの参照がオブジェクトに対応していません。
E0520246	［メッセージ］	A nonstatic data member may not be defined outside its class.
	［説明］	非静的データ・メンバはクラス外で定義できません。
E0520247	［メッセージ］	"*symbol*" has already been defined.
	［説明］	"*symbol*" はすでに定義されています。
E0520248	［メッセージ］	Pointer to reference is not allowed.
	［説明］	リファレンス型へのポインタ型は許されません。
E0520249	［メッセージ］	Reference to reference is not allowed.
	［説明］	リファレンス型へのリファレンス型は許されません。
E0520250	［メッセージ］	Reference to void is not allowed.
	［説明］	void 型へのリファレンス型は許されません。
E0520251	［メッセージ］	Array of reference is not allowed.
	［説明］	リファレンス型の配列は許されません。
E0520252	［メッセージ］	Reference " *名前* " requires an initializer.
	［説明］	リファレンス型の定義 " *名前* " には初期値が必要です。
E0520253	［メッセージ］	Expected a ",".
	［説明］	"," がありません。
E0520254	［メッセージ］	Type name is not allowed.
	［説明］	型名は許されていません。
E0520255	［メッセージ］	Type definition is not allowed.
	［説明］	型定義は許されていません。
E0520256	［メッセージ］	Invalid redeclaration of type name "*symbol*"
	［説明］	型名 "*symbol*" が不正に再宣言されています。
E0520257	［メッセージ］	const type "*symbol*" requires an initializer.
E0520258	［メッセージ］	"this" may only be used inside a nonstatic member function.
	［説明］	"this" が非静的メンバ関数以外で使われています。
E0520259	［メッセージ］	Constant value is not known.
	［説明］	定数値が不明です。

E0520260	［メッセージ］	Explicit type is missing ("int" assumed).
	［説明］	明示的な型がありません。"int" として扱います。
E0520262	［メッセージ］	Not a class or struct name.
	［説明］	基底クラスで指定されたクラスまたは構造体がありません。
E0520263	［メッセージ］	Duplicate base class name.
	［説明］	基底クラスを二重に指定しています。
E0520264	［メッセージ］	Invalid base class.
	［説明］	基底クラスが不正です。
E0520265	［メッセージ］	" *名前* " is inaccessible.
	［説明］	" *名前* " をアクセスすることはできません。
E0520266	［メッセージ］	" *名前* " is ambiguous.
	［説明］	指定された " *名前* " があいまいです。
E0520267	［メッセージ］	Old-style parameter list (anachronism).
	［説明］	古い仕様の引数リストです。
E0520268	［メッセージ］	Declaration may not appear after executable statement in block.
	［説明］	ブロック内で実行文の後に宣言を置けません。
E0520269	［メッセージ］	Conversion to inaccessible base class " *型* " is not allowed.
	［説明］	参照不可能な基底クラス " *型* " に変換できません。
E0520274	［メッセージ］	Improperly terminated macro invocation.
	［説明］	不適切に終了したマクロの呼び出しがあります。
E0520276	［メッセージ］	Name followed by "::" must be a class or namespace name.
	［説明］	"::" が続く名前は class 名または namespace 名でなければなりません。
E0520277	［メッセージ］	Invalid friend declaration.
	［説明］	フレンド宣言の指定が正しくありません。
E0520278	［メッセージ］	A constructor or destructor may not return a value.
	［説明］	コンストラクタやデストラクタはリターン値を持てません。
E0520279	［メッセージ］	Invalid destructor declaration.
	［説明］	デストラクタの宣言が正しくありません。
E0520280	［メッセージ］	Declaration of a member with the same name as its class.
	［説明］	クラス名と同じ名前のメンバ名を宣言しています。
E0520281	［メッセージ］	Global-scope qualifier (leading "::") is not allowed.
	［説明］	グローバルなスコープ決定演算子は許されません。
E0520282	［メッセージ］	The global scope has no *xxx*.
	［説明］	グローバルスコープは *xxx* を持ちません。
E0520283	［メッセージ］	Qualified name is not allowed.
	［説明］	修飾名は許されていません。

E0520284	［メッセージ］	NULL reference is not allowed.
	［説明］	NULL へのリファレンスは許されません。指定された通りに式を評価します。
E0520285	［メッセージ］	Initialization with "{...}" is not allowed for object of type "*type*".
	［説明］	{...} を伴った初期化は "*type*" 型のオブジェクトには許されていません。
E0520286	［メッセージ］	Base class " *型* " is ambiguous.
	［説明］	基底クラスの型があいまいです。
E0520287	［メッセージ］	Derived class *型 1* contains more than one instance of class *型 2*.
	［説明］	派生型が複数の同一クラス " *型* " を含みます。
E0520288	［メッセージ］	Cannot convert pointer to base class *型 2* to pointer to derived class *型 1* -- base class is virtual.
	［説明］	仮想基底クラス " *型 1*" のポインタ型を派生クラス " *型 2*" のポインタ型に変換することはできません。
E0520289	［メッセージ］	No instance of constructor *名前* matches the argument list.
	［説明］	コンストラクタ " *名前* " の引数が一致しません。
E0520290	［メッセージ］	Copy constructor for class *型* is ambiguous.
	［説明］	クラス " *型* " のコピーコンストラクタがあいまいです。
E0520291	［メッセージ］	No default constructor exists for class *型* .
	［説明］	クラス " *型* " のデフォルトコンストラクタは存在しません。
E0520292	［メッセージ］	*名前* is not a nonstatic data member or base class of class *型* .
	［説明］	" *名前* " が非静的データ・メンバまたは基底クラス " *型* " でありません。
E0520293	［メッセージ］	Indirect nonvirtual base class is not allowed.
	［説明］	仮想でない間接基底クラスは許されません。
E0520294	［メッセージ］	Invalid union member -- class *型* has a disallowed member function.
	［説明］	union メンバに指定できないクラス " *型* " のメンバ関数があります。
E0520296	［メッセージ］	Invalid use of non-lvalue array.
	［説明］	左辺値でない配列の不正な利用です。
E0520297	［メッセージ］	Expected an operator.
	［説明］	演算子が必要です。
E0520298	［メッセージ］	Inherited member is not allowed.
	［説明］	継承されたメンバを使用することはできません。
E0520299	［メッセージ］	Cannot determine which instance of *名前* is intended.
	［説明］	オーバーロード関数の " *名前* " を決定できません。
E0520300	［メッセージ］	A pointer to a bound function may only be used to call the function.
	［説明］	メンバ関数へのポインタを関数呼び出し以外に使用しています。
E0520301	［メッセージ］	typedef name has already been declared (with same type).
	［説明］	typedef 名はすでに同じ型で宣言されています。

E0520302	［メッセージ］	Symbol has already been defined.
	［説明］	symbol はすでに定義されています。
E0520304	［メッセージ］	No instance of *名前* matches the argument list.
	［説明］	関数 " *名前* " の引数が一致しません。
E0520305	［メッセージ］	Type definition is not allowed in function return type declaration.
	［説明］	関数の戻り値の宣言において型定義はできません。
E0520306	［メッセージ］	Default argument not at end of parameter list.
	［説明］	デフォルト引数の宣言がパラメータリストの最後ではありません。
E0520307	［メッセージ］	Redefinition of default argument.
	［説明］	デフォルト引数を再定義しています。
E0520308	［メッセージ］	More than one instance of *名前* matches the argument list:
	［説明］	引数リストが一致するためオーバーロード関数 " *名前* " があいまいです。
E0520309	［メッセージ］	More than one instance of constructor *名前* matches the argument list:
	［説明］	引数リストが一致するためコンストラクタ " *名前* " があいまいです。
E0520310	［メッセージ］	Default argument of type *型 1* is incompatible with parameter of type *型 2*.
	［説明］	デフォルト値の " *型 1* " が引数の " *型 2* " に合致しません。
E0520311	［メッセージ］	Cannot overload functions distinguished by return type alone.
	［説明］	リターン型が異なる関数をオーバーロードすることはできません。
E0520312	［メッセージ］	No suitable user-defined conversion from *型 1* to *型 2* exists.
	［説明］	適切な利用者定義変換 " *型 1* " から " *型 2* " が存在しません。
E0520313	［メッセージ］	Type qualifier is not allowed on this function.
	［説明］	関数に型限定子 (const,volatile) を指定することはできません。
E0520314	［メッセージ］	Only nonstatic member functions may be virtual.
	［説明］	静的メンバ関数に virtual を指定しています。
E0520315	［メッセージ］	The object has cv-qualifiers that are not compatible with the member function.
	［説明］	オブジェクトの型限定子 (const,volatile) がメンバ関数の型限定子と合致しません。
E0520316	［メッセージ］	Program too large to compile (too many virtual functions).
	［説明］	仮想関数の数が多すぎます。
E0520317	［メッセージ］	Return type is not identical to nor covariant with return type *型* of overridden virtual function *名前* .
	［説明］	仮想関数 " *名前* " のリターン型 " *型* " が異なります。
E0520318	［メッセージ］	Override of virtual *名前* is ambiguous.
	［説明］	仮想関数 " *名前* " の置き換えがあいまいです。
E0520319	［メッセージ］	Pure specifier ("= 0") allowed only on virtual functions.
	［説明］	純粋指定子 "=0" を仮想関数以外に指定しています。
E0520320	［メッセージ］	Badly-formed pure specifier (only "= 0" is allowed).
	［説明］	純粋指定子の形式が正しくありません。"=0" だけが許されます。

E0520321	［メッセージ］	Data member initializer is not allowed.
	［説明］	データ・メンバー初期化子は許されていません。
E0520322	［メッセージ］	Object of abstract class type *型* is not allowed:
	［説明］	抽象クラス " *型* " のオブジェクトは定義できません。
E0520323	［メッセージ］	function returning abstract class *型* is not allowed:
	［説明］	抽象クラス " *型* " を返す関数は定義できません。
E0520325	［メッセージ］	inline specifier allowed on function declarations only.
	［説明］	inline 指定子は関数宣言のみに利用できます。
E0520326	［メッセージ］	inline is not allowed.
	［説明］	inline は許されていません。
E0520327	［メッセージ］	Invalid storage class for an inline function.
	［説明］	inline 関数の記憶クラスが不正です。
E0520328	［メッセージ］	Invalid storage class for a class member.
	［説明］	クラスメンバの記憶クラスが不正です。
E0520329	［メッセージ］	Local class member *名前* requires a definition.
	［説明］	局所クラスメンバ " *名前* " の定義がありません。
E0520330	［メッセージ］	*名前* is inaccessible.
	［説明］	*名前*をアクセスできません。
E0520332	［メッセージ］	class *型* has no copy constructor to copy a const object.
	［説明］	クラス " *型* " に const 型オブジェクトをコピーするコピーコンストラクタがありません。
E0520333	［メッセージ］	Defining an implicitly declared member function is not allowed.
	［説明］	暗黙宣言されたメンバ関数を定義することはできません。
E0520334	［メッセージ］	class *型* has no suitable copy constructor.
	［説明］	クラス " *型* " に適切なコピーコンストラクタが存在しません。
E0520335	［メッセージ］	Linkage specification is not allowed.
	［説明］	リンケージの指定は許されていません。
E0520336	［メッセージ］	Unknown external linkage specification.
	［説明］	不明な外部リンケージ指定です。
E0520337	［メッセージ］	Linkage specification is incompatible with previous "*symbol*".
	［説明］	リンケージの指定がすでにある "*symbol*" と適合しません。
E0520338	［メッセージ］	More than one instance of overloaded function *名前* has "C" linkage.
	［説明］	C リンケージを持ったオーバーロード関数 " *名前* " が複数あります。
E0520339	［メッセージ］	class *型* has more than one default constructor.
	［説明］	クラス " *型* " は複数のデフォルトコンストラクタを持っています。

E0520340	［メッセージ］	Value copied to temporary, reference to temporary used.
	［説明］	値がローカルな領域にコピーされました。ローカルな領域への参照が使用されます。
E0520341	［メッセージ］	"operator *演算子*" must be a member function.
	［説明］	演算子関数 " *演算子* " はメンバ関数でなければなりません。
E0520342	［メッセージ］	Operator may not be a static member function.
	［説明］	静的メンバ関数の演算子関数は許されません。
E0520343	［メッセージ］	No arguments allowed on user-defined conversion.
	［説明］	利用者定義変換に引数は許されません。
E0520344	［メッセージ］	Too many parameters for this operator function.
	［説明］	演算子関数の引数の数が多すぎます。
E0520345	［メッセージ］	Too few parameters for this operator function.
	［説明］	演算子関数の引数の数が足りません。
E0520346	［メッセージ］	Nonmember operator requires a parameter with class type.
	［説明］	メンバ関数でない演算子関数はクラス型を引数に持つ必要があります。
E0520347	［メッセージ］	Default argument is not allowed.
	［説明］	デフォルト引数は許されていません。
E0520348	［メッセージ］	More than one user-defined conversion from *型 1* to *型 2* applies:
	［説明］	*型 1* から " *型 2*" への利用者定義型変換があいまいです。
E0520349	［メッセージ］	No operator *演算子* matches these operands.
	［説明］	演算子関数 " *演算子* " のオペランドが一致しません。
E0520350	［メッセージ］	More than one operator *演算子* matches these operands:
	［説明］	演算子関数 " *演算子* " のオペランドがあいまいです。
E0520351	［メッセージ］	First parameter of allocation function must be of type "size_t".
	［説明］	operator new の第 1 パラメータは size_t 型でなければなりません。
E0520352	［メッセージ］	Allocation function requires "void *" return type.
	［説明］	operator new のリターン型は void * 型でなければなりません。
E0520353	［メッセージ］	Deallocation function requires "void" return type.
	［説明］	operator delete のリターン型は void 型でなければなりません。
E0520354	［メッセージ］	First parameter of deallocation function must be of type "void *".
	［説明］	operator delete の第 1 パラメータは void * 型でなければなりません。
E0520356	［メッセージ］	Type must be an object type.
	［説明］	型はオブジェクト型でなければなりません。
E0520357	［メッセージ］	Base class *xxx* has already been initialized.
	［説明］	基底クラスはすでに初期化されています。
E0520358	［メッセージ］	Base class name required -- *xxx* assumed (anachronism).
	［説明］	名前はすでに初期化されています。

E0520359	［メッセージ］	Symbol has already been initialized.
	［説明］	symbol はすでに初期化されています。
E0520360	［メッセージ］	Name of member or base class is missing.
	［説明］	メンバ名または基底クラスに誤りがあります。
E0520363	［メッセージ］	Invalid anonymous union -- nonpublic member is not allowed.
	［説明］	無名 union のメンバが公開メンバでありません。
E0520364	［メッセージ］	Invalid anonymous union -- member function is not allowed.
	［説明］	無名 union にメンバ関数は許されません。
E0520365	［メッセージ］	Anonymous union at global or namespace scope must be declared static.
	［説明］	グローバルまたは namespace スコープの無名 union は static 宣言が必要です。
E0520366	［メッセージ］	Symbol provides no initializer for:
	［説明］	symbol が初期化に使用されました :
E0520367	［メッセージ］	Implicitly generated constructor for class *型* cannot initialize:
	［説明］	暗黙に生成されたクラス " *型* " のコンストラクタを初期化することはできません。
E0520369	［メッセージ］	*名前* has an uninitialized const or reference member.
	［説明］	*名前*の const またはリファレンスメンバが初期化されていません。
E0520371	［メッセージ］	class *型* has no assignment operator to copy a const object.
	［説明］	const オブジェクトをコピーするクラス " *型* " の代入演算子関数が定義されていません。
E0520372	［メッセージ］	class *型* has no suitable assignment operator.
	［説明］	クラス " *型* " に適当な代入演算が定義されていません。
E0520373	［メッセージ］	Ambiguous assignment operator for class *型* .
	［説明］	クラス " *型* " の代入演算子関数があいまいです。
E0520375	［メッセージ］	Declaration requires a typedef name.
	［説明］	宣言は typedef 名を必要とします。
E0520378	［メッセージ］	static is not allowed.
	［説明］	static は許されていません。
E0520380	［メッセージ］	Expression must have pointer-to-member type.
	［説明］	式はメンバへのポインタ型でなければなりません。
E0520384	［メッセージ］	No instance of overloaded *名前* matches the argument list.
	［説明］	オーバーロード関数 " *名前* " の引数リストが一致しません。
E0520386	［メッセージ］	No instance of *名前* matches the required type.
	［説明］	要求される型のオーバーロード関数 " *名前* " がありません。
E0520389	［メッセージ］	A cast to abstract class *型* is not allowed:
	［説明］	抽象クラス " *型* " へのキャストは許されません。
E0520390	［メッセージ］	Function "main" may not be called or have its address taken.
	［説明］	関数 "main" の呼び出しおよびアドレスの取得は禁止されています。

E0520391	［メッセージ］	A new-initializer may not be specified for an array.
	［説明］	配列を new によって初期化することはできません。
E0520392	［メッセージ］	Member function *名前* may not be redeclared outside its class.
	［説明］	メンバ関数 " *名前* " がクラスの外側で再宣言されました。
E0520393	［メッセージ］	Pointer to incomplete class type is not allowed.
	［説明］	不完全クラスへのポインタ型は許されません。
E0520394	［メッセージ］	Reference to local variable of enclosing function is not allowed.
	［説明］	ローカルクラスを囲む関数の局所変数へのリファレンスは許されません。
E0520397	［メッセージ］	Implicitly generated assignment operator cannot copy:
	［説明］	暗黙に生成された代入演算子関数がオブジェクトを正しくコピーすることができません。
E0520401	［メッセージ］	Destructor for base class *型* is not virtual.
	［説明］	基底クラス " *型* " のデストラクタが virtual でありません。
E0520403	［メッセージ］	Invalid redeclaration of member "symbol".
	［説明］	メンバ "symbol" の再宣言は無効です。
E0520404	［メッセージ］	Function "main" may not be declared inline.
	［説明］	関数 "main" は inline 宣言できません。
E0520405	［メッセージ］	Member function with the same name as its class must be a constructor.
	［説明］	クラス名と同じ名前のメンバ関数はコンストラクタでなければなりません。
E0520407	［メッセージ］	A destructor may not have parameters.
	［説明］	デストラクタは引数を持つことができません。
E0520408	［メッセージ］	Copy constructor for class *型 1* may not have a parameter of type *型 2*.
	［説明］	クラス " *型 1* " のコピーコンストラクタは " *型 2* " の引数を持つことはできません。
E0520409	［メッセージ］	*symbol* returns incomplete type "*type*"
	［説明］	*symbol* は不完全型 "*type*" を返します。
E0520410	［メッセージ］	Protected *名前* is not accessible through a *型* pointer or object.
	［説明］	限定公開名 " *名前* " は " *型* " へのポインタやオブジェクトを経由してアクセスすることはできません。
E0520411	［メッセージ］	A parameter is not allowed.
	［説明］	引数は許されていません。
E0520413	［メッセージ］	No suitable conversion function from *型 1* to *型 2* exists.
	［説明］	*型 1* から " *型 2* " への適切な変換関数が存在しません。
E0520415	［メッセージ］	No suitable constructor exists to convert from *型 1* to *型 2*.
	［説明］	*型 1* から " *型 2* " へ変換する適切なコンストラクタが存在しません。
E0520416	［メッセージ］	More than one constructor applies to convert from *型 1* to *型 2*:
	［説明］	*型 1* から " *型 2* " へ変換するコンストラクタがあいまいです。

E0520417	［メッセージ］	More than one conversion function from *型 1* to *型 2* applies:
	［説明］	*型 1* から " *型 2*" への変換関数があいまいです。
E0520418	［メッセージ］	More than one conversion function from *型* to a built-in type applies:
	［説明］	*型*から組み込み型への変換関数があいまいです。
E0520424	［メッセージ］	A constructor or destructor may not have its address taken.
	［説明］	コンストラクタ、またはデストラクタのアドレスを参照することはできません。
E0520427	［メッセージ］	Qualified name is not allowed in member declaration.
	［説明］	限定名をメンバ宣言の中で使用できません。
E0520429	［メッセージ］	The size of an array in "new" must be non-negative.
	［説明］	new で指定された配列のサイズに負の値は許されません。
E0520432	［メッセージ］	enum declaration is not allowed.
	［説明］	enum 宣言は許されていません。
E0520433	［メッセージ］	Qualifiers dropped in binding reference of type *型 1* to initializer of type *型 2*.
	［説明］	const/volatile 限定の型 " *型 2*" が参照型 " *型 1*" の初期値に指定されました。
E0520434	［メッセージ］	A reference of type *型1* (not const-qualified) cannot be initialized with a value of type *型 2*.
	［説明］	const 型修飾されない型 " *型 1*" へのリファレンスを " *型 2*" の値で初期化できません。
E0520435	［メッセージ］	A pointer to function may not be deleted.
	［説明］	関数へのポインタを削除することはできません。
E0520436	［メッセージ］	Conversion function must be a nonstatic member function.
	［説明］	変換関数は非静的メンバ関数でなければなりません。
E0520437	［メッセージ］	Template declaration is not allowed here.
	［説明］	このスコープ内でテンプレート宣言は許されません。
E0520438	［メッセージ］	Expected a "<".
	［説明］	< が必要です。
E0520439	［メッセージ］	Expected a ">".
	［説明］	> が必要です。
E0520440	［メッセージ］	Template parameter declaration is missing.
	［説明］	テンプレートの引数宣言が正しくありません。
E0520441	［メッセージ］	Argument list for " *名前* " is missing.
	［説明］	テンプレート " *名前* " の実引数リストが正しくありません。
E0520442	［メッセージ］	Too few arguments for " *名前* ".
	［説明］	テンプレート " *名前* " の実引数が足りません。
E0520443	［メッセージ］	Too many arguments for "symbol".
	［説明］	テンプレートの実引数が多すぎます。

E0520445	［メッセージ］	" *名前 1*" is not used in declaring the parameter types of " *名前 2*".
	［説明］	テンプレート " *名前 2*" の引数 " *名前 1*" が使用されません。
E0520449	［メッセージ］	More than one instance of *名前* matches the required type.
	［説明］	オーバーロード関数 " *名前* " があいまいです。
E0520450	［メッセージ］	The type "long long" is nonstandard.
	［説明］	型 "long long" は標準ではありません。
E0520451	［メッセージ］	Omission of "class" is nonstandard.
	［説明］	"class" のない friend 宣言は標準形式ではありません。
E0520452	［メッセージ］	Return type may not be specified on a conversion function.
	［説明］	変換関数のリターン型が指定されていません。
E0520456	［メッセージ］	Excessive recursion at instantiation of *名前* .
	［説明］	テンプレート " *名前* " のインスタンスが再帰的に生成されます。
E0520457	［メッセージ］	*名前* is not a function or static data member.
	［説明］	*名前*が関数または静的データ・メンバではありません。
E0520458	［メッセージ］	Argument of type *型 1* is incompatible with template parameter of type *型 2*.
	［説明］	実引数の型 " *型 1*" がテンプレートの引数 " *型 2*" に合致しません。
E0520459	［メッセージ］	Initialization requiring a temporary or conversion is not allowed.
	［説明］	初期化にテンポラリや変換を要求することは許されません。
E0520460	［メッセージ］	declaration of *xxx* hides function parameter.
	［説明］	*xxx* の宣言は関数引数を隠します。
E0520461	［メッセージ］	Initial value of reference to non-const must be an lvalue.
	［説明］	const 型を持たないリファレンスの初期値は左辺値でなければなりません。
E0520463	［メッセージ］	"template" is not allowed.
	［説明］	template 指定は許されません。
E0520464	［メッセージ］	*型* is not a class template.
	［説明］	*型*がクラステンプレートではありません。
E0520466	［メッセージ］	"main" is not a valid name for a function template.
	［説明］	main は関数テンプレートの名前に使用できません。
E0520467	［メッセージ］	Invalid reference to *名前* (union/nonunion mismatch).
	［説明］	*名前*の参照が不正です。
E0520468	［メッセージ］	A template argument may not reference a local type.
	［説明］	テンプレートの実引数はローカルな型を参照できません。
E0520469	［メッセージ］	Tag kind of *xxx* is incompatible with declaration of "symbol".
	［説明］	*xxx* のタグの種類は、"symbol" の宣言と一致しません。
E0520470	［メッセージ］	The global scope has no tag named *xxx*.
	［説明］	グローバルスコープは *xxx* というタグを持っていません。

E0520471	［メッセージ］	symbol has no tag member named *xxx*.
	［説明］	symbol は *xxx* というタグメンバを持ちません。
E0520473	［メッセージ］	*名前* may be used only in pointer-to-member declaration.
	［説明］	typedef 名 " *名前* " はメンバへのポインタ型の宣言の中で使用されなければなりません。
E0520475	［メッセージ］	A template argument may not reference a non-external entity.
	［説明］	テンプレートの実引数は外部名以外を参照できません。
E0520476	［メッセージ］	Name followed by "::~" must be a class name or a type name.
	［説明］	"::~" に続く名前はクラス名または型名でなければなりません。
E0520477	［メッセージ］	Destructor name does not match name of class *型* .
	［説明］	クラス名 " *型* " とデストラクタ名が合致しません。
E0520478	［メッセージ］	Type used as destructor name does not match type *型* .
	［説明］	デストラクタ名で使われた型と " *型* " が合致しません。
E0520481	［メッセージ］	Invalid storage class for a template declaration.
	［説明］	テンプレート宣言の記憶クラス指定が正しくありません。
E0520484	［メッセージ］	Invalid explicit instantiation declaration.
	［説明］	テンプレートの実引数が不正です。
E0520485	［メッセージ］	*名前* is not an entity that can be instantiated.
	［説明］	テンプレート " *名前* " を実体化できません。
E0520486	［メッセージ］	Compiler generated *名前* cannot be explicitly instantiated.
	［説明］	コンパイラが生成した関数を実体化することはできません。
E0520487	［メッセージ］	Inline *名前* cannot be explicitly instantiated.
	［説明］	インライン関数 " *名前* " を実体化することはできません。
E0520489	［メッセージ］	*名前* cannot be instantiated -- no template definition was supplied.
	［説明］	テンプレート定義がないため " *名前* " を実体化することはできません。
E0520490	［メッセージ］	*名前* cannot be instantiated -- it has been explicitly specialized.
	［説明］	*名前*を実体化することはできません。
E0520493	［メッセージ］	No instance of *名前* matches the specified type.
	［説明］	オーバーロード関数 " *名前* " と指定された型が合致しません。
E0520494	［メッセージ］	Declaring a void parameter list with a typedef is nonstandard.
	［説明］	typedef を伴う void の引数リストの宣言は標準でありません。
E0520496	［メッセージ］	Template parameter *名前* may not be redeclared in this scope.
	［説明］	テンプレート引数 " *名前* " がスコープ内で再宣言されています。
E0520498	［メッセージ］	Template argument list must match the parameter list.
	［説明］	テンプレート実引数と仮引数が合致しません。
E0520500	［メッセージ］	Extra parameter of postfix "operator *xxx*" must be of type "int".
	［説明］	後置演算関数の第 2 パラメータの型は int 型でなければなりません。

E0520501	［メッセージ］	An operator name must be declared as a function.
	［説明］	演算子名は関数として宣言しなければなりません。
E0520502	［メッセージ］	Operator name is not allowed.
	［説明］	演算子名は許されません。
E0520503	［メッセージ］	*名前* cannot be specialized in the current scope.
	［説明］	スコープ内で " *名前* " があいまいです。
E0520504	［メッセージ］	Nonstandard form for taking the address of a member function.
	［説明］	メンバ関数のアドレスを取得するのは標準形式ではありません。
E0520505	［メッセージ］	Too few template parameters -- does not match previous declaration.
	［説明］	テンプレートの引数が足りません。
E0520506	［メッセージ］	Too many template parameters -- does not match previous declaration.
	［説明］	テンプレートの引数が多すぎます。
E0520507	［メッセージ］	Function template for operator delete(void *) is not allowed.
	［説明］	operator delete(void *) の関数テンプレートは許されません。
E0520508	［メッセージ］	class template and template parameter may not have the same name.
	［説明］	クラステンプレートとテンプレートの引数が同じ名前です。
E0520510	［メッセージ］	A template argument may not reference an unnamed type.
	［説明］	テンプレートの実引数が名前付けされていない型を参照しています。
E0520511	［メッセージ］	Enumerated type is not allowed.
	［説明］	列挙型は許されていません。
E0520513	［メッセージ］	A value of type " *型名 1* " cannot be assigned to an entity of type " *型名 2* ".
	［説明］	型 " *型名 1* " の値は型 " *型名 2* " の実体として代入できません。
E0520515	［メッセージ］	Cannot convert to incomplete class *型* .
	［説明］	不完全型 " *型* " への型変換はできません。
E0520516	［メッセージ］	const object requires an initializer.
	［説明］	const のオブジェクトは初期化が必要です。
E0520517	［メッセージ］	Object has an uninitialized const or reference member.
	［説明］	オブジェクトが未初期化の const 型メンバあるいはリファレンス型メンバを持ちます。
E0520518	［メッセージ］	Nonstandard preprocessing directive.
	［説明］	標準でない前処理指令です。
E0520519	［メッセージ］	*名前* may not have a template argument list.
	［説明］	*名前*はテンプレート実引数を持つことができません。
E0520520	［メッセージ］	Initialization with "{...}" expected for aggregate object.
	［説明］	集成体は "{...}" により初期化してください。
E0520521	［メッセージ］	Pointer-to-member selection class types are incompatible (*型 1* and *型 2*).
	［説明］	メンバへのポインタ型のクラスの型が " *型 1* " と " *型 2* " で合致しません。

E0520525	［メッセージ］	A dependent statement may not be a declaration.
	［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］	A parameter may not have void type.
	［説明］	引数は void 型を持てません。
E0520529	［メッセージ］	This operator is not allowed in a template argument expression.
	［説明］	テンプレートの実引数式に指定された演算は許されません。
E0520530	［メッセージ］	Try block requires at least one handler/
	［説明］	try 文に対応する catch 文がありません。
E0520531	［メッセージ］	Handler requires an exception declaration.
	［説明］	catch 文の (...) には例外宣言が必要です。
E0520532	［メッセージ］	Handler is masked by default handler.
	［説明］	ハンドラはデフォルトのハンドラにマスクされました。
E0520536	［メッセージ］	Exception specification is incompatible with that of previous *名前* .
	［説明］	例外処理指定が前の指定 " *名前* " と合致しません。
E0520540	［メッセージ］	Support for exception handling is disabled.
	［説明］	例外処理を行うオプション (exception) が指定されていません。
E0520543	［メッセージ］	Non-arithmetic operation not allowed in nontype template argument.
	［説明］	対応するテンプレートの実引数に非算術型変換は許されません。
E0520544	［メッセージ］	Use of a local type to declare a nonlocal variable.
	［説明］	ローカルでない変数の宣言にローカルな型が使用されました。
E0520545	［メッセージ］	Use of a local type to declare a function.
	［説明］	ローカルな型が関数宣言に使用されました。
E0520546	［メッセージ］	Transfer of control bypasses initialization of:
	［説明］	初期化されないパスがあります :
E0520548	［メッセージ］	Transfer of control into an exception handler.
	［説明］	例外ハンドラ処理が実行されます。
E0520551	［メッセージ］	symbol cannot be defined in the current scope.
	［説明］	symbol はこのスコープで定義できません。
E0520555	［メッセージ］	Tag kind of *名前* is incompatible with template parameter of type *型* .
	［説明］	タグ " *名前* " の種類とテンプレートの引数の " *型* " が合致しません。
E0520556	［メッセージ］	Function template for operator new(size_t) is not allowed.
	［説明］	operator new(size_t) の関数テンプレートは許されません。
E0520558	［メッセージ］	Pointer to member of type "*type*" is not allowed.
	［説明］	型 "*type*" のメンバへのポインタは許されていません。
E0520559	［メッセージ］	Tointer to member of type *型* is not allowed.
	［説明］	メンバへのポインタ型 " *型* " が誤っています。

E0520560	［メッセージ］	symbol is reserved for future use as a keyword.
	［説明］	symbol は将来のために予約されたキーワードです。
E0520561	［メッセージ］	Invalid macro definition:
	［説明］	無効なマクロ定義です :
E0520562	［メッセージ］	Invalid macro undefinition:
	［説明］	無効なマクロ定義の取り消しです :
E0520598	［メッセージ］	A template parameter may not have void type.
	［説明］	テンプレートの引数に void 型は指定できません。
E0520599	［メッセージ］	Excessive recursive instantiation of *名前* due to instantiate-all mode.
	［説明］	instantiate-all モードの指定によってテンプレート " *名前* " のインスタンスが再帰的に生成されます。
E0520601	［メッセージ］	A throw expression may not have void type.
	［説明］	throw 式に void 型は指定できません。
E0520603	［メッセージ］	Parameter of abstract class type *型* is not allowed:
	［説明］	抽象クラス " *型* " の引数は許されません。
E0520604	［メッセージ］	Array of abstract class *型* is not allowed:
	［説明］	抽象クラス " *型* " の配列は許されません。
E0520605	［メッセージ］	Floating-point template parameter is nonstandard.
	［説明］	浮動小数点のテンプレートパラメータは標準形式ではありません。
E0520606	［メッセージ］	This pragma must immediately precede a declaration.
	［説明］	この pragma は宣言の直前でなければなりません。
E0520607	［メッセージ］	This pragma must immediately precede a statement.
	［説明］	この pragma は式の直前に記述しなければいけません。
E0520608	［メッセージ］	This pragma must immediately precede a declaration or statement.
	［説明］	この pragma は宣言または式の直前に記述しなければいけません。
E0520609	［メッセージ］	This kind of pragma may not be used here.
	［説明］	この種類の pragma はここで使用してはいけません。
E0520612	［メッセージ］	Specific definition of inline template function must precede its first use.
	［説明］	インライン指定されたテンプレート関数は呼び出しの前に定義しなければなりません。
E0520615	［メッセージ］	Parameter type involves pointer to array of unknown bound.
	［説明］	引数の型は境界が不明な配列へのポインタを含みます。
E0520616	［メッセージ］	Parameter type involves reference to array of unknown bound.
	［説明］	引数の型に要素数の指定がない配列への参照が含まれています。
E0520618	［メッセージ］	Struct or union declares no named members.
	［説明］	構造体か共用体に名前のないメンバがあります。

E0520619	[メッセージ]	Nonstandard unnamed field.
	[説明]	名前のないフィールドは標準ではありません。
E0520620	[メッセージ]	Nonstandard unnamed member.
	[説明]	名前のないメンバは標準ではありません。
E0520643	[メッセージ]	restrict is not allowed.
	[説明]	restrict は許されていません。
E0520644	[メッセージ]	A pointer or reference to function type may not be qualified by "restrict".
	[説明]	関数へのポインタまたは参照型は ”restrict” によって修飾してはいけません。
E0520647	[メッセージ]	Conflicting calling convention modifiers.
	[説明]	呼び出し規則変更が競合しています。
E0520651	[メッセージ]	A calling convention may not be followed by a nested declarator.
	[説明]	呼び出し規則はネストされた宣言子に続けることはできません。
E0520654	[メッセージ]	Declaration modifiers are incompatible with previous declaration.
	[説明]	宣言修飾子が前の宣言と合致しません。
E0520656	[メッセージ]	Transfer of control into a try block.
	[説明]	外側のブロックから try ブロックに制御が移ります。
E0520658	[メッセージ]	Closing brace of template definition not found.
	[説明]	テンプレート定義の閉じ括弧がありません。
E0520660	[メッセージ]	Invalid packing alignment value.
	[説明]	パッキング値が不正です。
E0520661	[メッセージ]	Expected an integer constant.
	[説明]	整数定数がありません。
E0520663	[メッセージ]	Invalid source file identifier string.
	[説明]	ソース・ファイル識別文字列が不正です。
E0520664	[メッセージ]	A class template cannot be defined in a friend declaration.
	[説明]	フレンド宣言内でクラステンプレートを定義することはできません。
E0520665	[メッセージ]	asm is not allowed.
	[説明]	asm は許されていません。
E0520666	[メッセージ]	asm must be used with a function definition.
	[説明]	asm は関数定義とともに利用される必要があります。
E0520667	[メッセージ]	asm function is nonstandard.
	[説明]	asm 関数は標準ではありません。
E0520668	[メッセージ]	Ellipsis with no explicit parameters is nonstandard.
	[説明]	明示的な引数のない省略記号は標準ではありません。
E0520669	[メッセージ]	&... is nonstandard.
	[説明]	&... は標準ではありません。

E0520670	［メッセージ］	invalid use of "&...".
	［説明］	"&..." の使い方が不正です。
E0520673	［メッセージ］	A reference of type *型 1* cannot be initialized with a value of type *型 2*.
	［説明］	const/volatile 型 " *型 1*" のリファレンスは " *型 2*" の値で初期化できません。
E0520674	［メッセージ］	Initial value of reference to const volatile must be an lvalue.
	［説明］	const/volatile 型のリファレンスの初期値は左辺値でなければなりません。
E0520676	［メッセージ］	Using out-of-scope declaration of *type* "*symbol*" (declared at line *number*).
E0520691	［メッセージ］	*xxx*, required for copy that was eliminated, is inaccessible.
	［説明］	コピーコンストラクタにアクセスできません。
E0520692	［メッセージ］	*xxx* required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
	［説明］	コピーコンストラクタを呼び出すことができません。
E0520693	［メッセージ］	<typeinfo> must be included before typeid is used.
	［説明］	typeid を使うためには <typeinfo> をインクルードしなければなりません。
E0520694	［メッセージ］	*xxx* cannot cast away const or other type qualifiers.
	［説明］	" *名前*" のキャストで const などの属性を取り除くことはできません。
E0520695	［メッセージ］	The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *.
	［説明］	dynamic_cast の型は完全クラス型へのポインタ型またはリファレンス型か void * 型でなければなりません。
E0520696	［メッセージ］	The operand of a pointer dynamic_cast must be a pointer to a complete class type.
	［説明］	dynamic_cast ポインタのオペランドは完全クラス型へのポインタ型でなければなりません。
E0520697	［メッセージ］	The operand of a reference dynamic_cast must be an lvalue of a complete class type.
	［説明］	dynamic_cast のリファレンスのオペランドは完全クラス型の左辺値でなければなりません。
E0520698	［メッセージ］	The operand of a runtime dynamic_cast must have a polymorphic class type.
	［説明］	実行時 dynamic_cast のオペランドはポリモフィックなクラス型でなければなりません。
E0520701	［メッセージ］	An array type is not allowed here.
	［説明］	配列型はここでは許されません。
E0520702	［メッセージ］	Expected an "=".
	［説明］	"=" がありません。
E0520703	［メッセージ］	Expected a declarator in condition declaration.
	［説明］	コンディション宣言に宣言子がありません。
E0520704	［メッセージ］	*xxx*, declared in condition, may not be redeclared in this scope.
	［説明］	このスコープ内で " 名前 " を再宣言することはできません。

E0520705	［メッセージ］	Default template arguments are not allowed for function templates.
	［説明］	関数テンプレートにデフォルトの実引数を指定することはできません。
E0520706	［メッセージ］	Expected a "," or ">".
	［説明］	"," か ">" がありません。
E0520707	［メッセージ］	Expected a template parameter list.
	［説明］	テンプレートの引数リストが必要です。
E0520709	［メッセージ］	bool type is not allowed.
	［説明］	bool 型の値をデクリメントすることはできません。
E0520710	［メッセージ］	Offset of base class *名前 1* within class *名前 2* is too large.
	［説明］	クラス " *名前 2*" 内の基底クラス " *名前 1*" のサイズが大きすぎます。
E0520711	［メッセージ］	Expression must have bool type (or be convertible to bool).
	［説明］	式の型は bool 型か bool 型へ変換可能な型でなければなりません。
E0520717	［メッセージ］	The type in a const_cast must be a pointer, reference, or pointer to member to an object type.
	［説明］	const_cast の型はポインタ型、リファレンス型またはメンバへのポインタ型でなければなりません。
E0520718	［メッセージ］	A const_cast can only adjust type qualifiers; it cannot change the underlying type.
	［説明］	const_cast は const/volatile 以外の型を調整することはできません。
E0520719	［メッセージ］	mutable is not allowed.
	［説明］	mutable の指定は許されません。
E0520724	［メッセージ］	namespace definition is not allowed.
	［説明］	namespace の定義はファイルスコープまたは namespace スコープ内で許されます。
E0520725	［メッセージ］	name must be a namespace name.
	［説明］	namespace の名前が正しくありません。
E0520726	［メッセージ］	namespace alias definition is not allowed.
	［説明］	namespace の別名定義はここでは許されません。
E0520727	［メッセージ］	namespace-qualified name is required.
	［説明］	namespace の限定名が要求されます。
E0520728	［メッセージ］	A namespace name is not allowed.
	［説明］	namespace 名は許されません。
E0520730	［メッセージ］	*名前* is not a class template.
	［説明］	" *名前*" はクラステンプレートのメンバではありません。
E0520731	［メッセージ］	Array with incomplete element type is nonstandard.
	［説明］	不完全型の配列は標準ではありません。
E0520732	［メッセージ］	Allocation operator may not be declared in a namespace.
	［説明］	operator new 関数が namespace 内で宣言されています。

E0520733	［メッセージ］	Deallocation operator may not be declared in a namespace.
	［説明］	operator delete 関数が namespace 内で宣言されています。
E0520734	［メッセージ］	*名前 1* conflicts with using-declaration of *名前 2*.
	［説明］	名前 " *名前 1*" が using 宣言名 " *名前 2*" と衝突します。
E0520735	［メッセージ］	using-declaration of *名前 1* conflicts with *名前 2*.
	［説明］	using 宣言の名前が衝突します。
E0520742	［メッセージ］	symbol has no actual member *xxx*.
	［説明］	symbol は *xxx* の実際のメンバではありません。
E0520749	［メッセージ］	A type qualifier is not allowed.
	［説明］	型指定子は許されていません。
E0520750	［メッセージ］	*名前* was used before its template was declared.
	［説明］	" *名前*" はテンプレートが宣言される前に使われました。
E0520751	［メッセージ］	Static and nonstatic member functions with same parameter types cannot be overloaded.
	［説明］	同じ引数の型を持つ静的メンバ関数と非静的メンバ関数はオーバーロードすることはできません。
E0520752	［メッセージ］	No prior declaration of "symbol".
	［説明］	symbol の宣言がありません。
E0520753	［メッセージ］	A template-id is not allowed.
	［説明］	ここではテンプレート (template 名 <template 実引数 >) は許されません。
E0520754	［メッセージ］	A class-qualified name is not allowed.
	［説明］	ここではクラス限定名は許されません。
E0520755	［メッセージ］	symbol may not be redeclared in the current scope.
	［説明］	symbol はこのスコープで再宣言できません。
E0520756	［メッセージ］	Qualified name is not allowed in namespace member declaration.
	［説明］	namespace メンバの宣言で指定された限定名は許されません。
E0520757	［メッセージ］	symbol is not a type name.
	［説明］	symbol は型名ではありません。
E0520758	［メッセージ］	Explicit instantiation is not allowed in the current scope.
	［説明］	現在のスコープ範囲でインスタンスを明示的に生成することはできません。
E0520759	［メッセージ］	*シンボル* cannot be explicitly instantiated in the current scope.
	［説明］	*シンボル*は現在のスコープで明示的にインスタンス化できません。
E0520761	［メッセージ］	typename may only be used within a template.
	［説明］	typename キーワードはテンプレート内でのみ使用できます。
E0520765	［メッセージ］	Nonstandard character at start of object-like macro definition.
	［説明］	標準でないキャラクタがオブジェクト風なマクロ定義の最初に指定されました。

E0520766	［メッセージ］	Exception specification for virtual *名前 1* is incompatible with that of overridden *名前 2*.
	［説明］	仮想関数の例外指定 " *名前 1*" が " *名前 2*" に合致しません。
E0520767	［メッセージ］	Conversion from pointer to smaller integer.
E0520768	［メッセージ］	Exception specification for implicitly declared virtual *名前 1* is incompatible with that of overridden *名前 2*.
	［説明］	コンパイラが生成する暗黙の仮想関数 " *名前 1*" の例外指定が " *名前 2*" に合致しません。
E0520769	［メッセージ］	*名前 1*, implicitly called from *名前 2*, is ambiguous.
	［説明］	operator delete の呼び出しがあいまいです。
E0520771	［メッセージ］	"explicit" is not allowed.
	［説明］	explicit はクラス宣言内のコンストラクタにのみ指定できます。
E0520772	［メッセージ］	Declaration conflicts with *xxx* (reserved class name).
	［説明］	予約されたクラス名 type_info と衝突します。
E0520773	［メッセージ］	Only "()" is allowed as initializer for array "symbol".
	［説明］	配列 "symbol" の初期化子として "()" のみが許可されています。
E0520774	［メッセージ］	"virtual" is not allowed in a function template declaration.
	［説明］	関数テンプレートに virtual 指定はできません。
E0520775	［メッセージ］	Invalid anonymous union -- class member template is not allowed.
	［説明］	無名 union の指定が正しくありません。
E0520776	［メッセージ］	Template nesting depth does not match the previous declaration of %n.
	［説明］	テンプレートのパラメータのネストが前の宣言 " 名前 " と合致しません。
E0520777	［メッセージ］	This declaration cannot have multiple "template <...>" clauses.
	［説明］	この宣言に複数のテンプレート宣言はできません。
E0520779	［メッセージ］	*xxx*, declared in for-loop initialization, may not be redeclared in this scope.
	［説明］	for 文の初期化式で宣言された " 名前 " をこのスコープ内で再宣言できません。
E0520782	［メッセージ］	Definition of virtual *名前* is required here.
	［説明］	仮想関数の定義 " *名前*" が必要です。
E0520784	［メッセージ］	A storage class is not allowed in a friend declaration.
	［説明］	フレンド宣言に記憶クラスを指定することはできません。
E0520785	［メッセージ］	Template parameter list for *名前* is not allowed in this declaration.
	［説明］	この宣言内に " *名前*" のテンプレートの引数並びは許されません。
E0520786	［メッセージ］	*名前* is not a valid member class or function template.
	［説明］	" *名前*" は有効なメンバまたは関数テンプレートではありません。
E0520787	［メッセージ］	Not a valid member class or function template declaration.
	［説明］	有効なメンバまたは関数テンプレート宣言ではありません。

E0520788	［メッセージ］	A template declaration containing a template parameter list may not be followed by an explicit specialization declaration.
	［説明］	テンプレート関数の定義の後にテンプレート引数並びを含むテンプレート宣言は指定できません。
E0520789	［メッセージ］	Explicit specialization of *名前 1* must precede the first use of *名前 2*.
	［説明］	明示的なテンプレートの実体の定義 " *名前 1*" は最初のテンプレート " *名前 2*" を使用する前になければなりません。
E0520790	［メッセージ］	Explicit specialization is not allowed in the current scope.
	［説明］	明示的なテンプレートの実体の定義はこのスコープでは許されません。
E0520791	［メッセージ］	Partial specialization of *名前* is not allowed.
	［説明］	テンプレート " *名前* " の部分的な定義は許されません。
E0520792	［メッセージ］	*名前* is not an entity that can be explicitly specialized.
	［説明］	" *名前* " はテンプレートのインスタンスではありません。
E0520793	［メッセージ］	Explicit specialization of %n must precede its first use.
	［説明］	明示的なテンプレートの実体 " 名前 " の定義は最初の使用より前になければなりません。
E0520795	［メッセージ］	Specializing *名前* requires "template<>" syntax.
	［説明］	" *名前* " のテンプレートの実体定義は tempalte<> 形式が要求されます。
E0520799	［メッセージ］	Specializing *シンボル* without "template<>" syntax is nonstandard.
	［説明］	“template<>” なしで " *シンボル* " を特殊化するのは標準形式ではありません。
E0520800	［メッセージ］	This declaration may not have extern "C" linkage.
	［説明］	この宣言は extern "C" リンケージを持つことはできません。
E0520801	［メッセージ］	*名前* is not a class or function template name in the current scope.
	［説明］	" *名前* " はこのスコープ内ではクラステンプレートまたは関数テンプレートではありません。
E0520803	［メッセージ］	Specifying a default argument when redeclaring an already referenced function template is not allowed.
	［説明］	すでに参照された関数テンプレートを再宣言するときにデフォルト引数を指定しています。
E0520804	［メッセージ］	Cannot convert pointer to member of base class *型2* to pointer to member of derived class *型 1* -- base class is virtual.
	［説明］	仮想基底クラス " *型 1*" のメンバポインタを派生クラス " *型 2*" のメンバポインタに変換することはできません。
E0520805	［メッセージ］	Exception specification is incompatible with that of *名前* .
	［説明］	throw 例外指定は " *名前* " の例外指定と合致しません。
E0520807	［メッセージ］	Unexpected end of default argument expression.
	［説明］	デフォルト引数式が正しくありません。
E0520808	［メッセージ］	Default-initialization of reference is not allowed.
	［説明］	リファレンス型のデフォルトの初期化は許されません。

E0520809	［メッセージ］	Uninitialized "symbol" has a const member.
	［説明］	初期化されていない "symbol" が const メンバを持っています。
E0520810	［メッセージ］	Uninitialized base class *型* has a const member.
	［説明］	未初期化の基底クラス " *型* " が const 型メンバを持ちます。
E0520811	［メッセージ］	const *名前* requires an initializer -- class *型* has no explicitly declared default constructor.
	［説明］	const 型の " *名前* " には初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。
E0520812	［メッセージ］	Const object requires an initializer -- class *型* has no explicitly declared default constructor.
	［説明］	const 型オブジェクトには初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。
E0520816	［メッセージ］	In a function definition a type qualifier on a "void" return type is not allowed.
	［説明］	関数定義において "void" の返却型を修飾することはできません。
E0520817	［メッセージ］	Static data member declaration is not allowed in this class.
	［説明］	局所クラスは静的データ・メンバを持つことはできません。
E0520818	［メッセージ］	Template instantiation resulted in an invalid function declaration.
	［説明］	テンプレートで実体化された関数宣言が正しくありません。
E0520819	［メッセージ］	... is not allowed.
	［説明］	... は許されていません。
E0520822	［メッセージ］	Invalid destructor name for type *型* .
	［説明］	" *型* " のデストラクタ名が正しくありません。
E0520824	［メッセージ］	Destructor reference is ambiguous -- both *名前１* and *名前２* could be used.
	［説明］	" *名前１* " と " *名前２* " が使われました。デストラクタの参照があいまいです。
E0520827	［メッセージ］	Only one member of a union may be specified in a constructor initializer list.
	［説明］	共用体の一つのメンバのみをコンストラクタの初期化で指定できます。
E0520828	［メッセージ］	Support for "new[]" and "delete[]" is disabled.
	［説明］	new[] と delete[] はサポートされていません。
E0520832	［メッセージ］	No appropriate operator delete is visible.
	［説明］	適当な operator delete 関数が見つかりません。
E0520833	［メッセージ］	Pointer or reference to incomplete type is not allowed.
	［説明］	不完全型へのポインタまたはリファレンス型は許されません。
E0520834	［メッセージ］	Invalid partial specialization -- *名前* is already fully specialized.
	［説明］	すでに特別化された " *名前* " を部分特別化しています。
E0520835	［メッセージ］	Incompatible exception specifications.
	［説明］	例外指定の型が合致しません。
E0520840	［メッセージ］	A template argument list is not allowed in a declaration of a primary template.
	［説明］	プライマリテンプレート宣言にテンプレート実引数は指定できません。

E0520841	［メッセージ］	Partial specializations may not have default template arguments.
	［説明］	部分特別化テンプレートはデフォルトのテンプレート引数を持つことはできません。
E0520842	［メッセージ］	*名前 1* is not used in or cannot be deduced from the template argument list of *名前 2*.
	［説明］	部分特別化テンプレート " *名前 1*" は " *名前 2*" のテンプレート実引数に使用されません。
E0520844	［メッセージ］	The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter.
	［説明］	部分特別化テンプレートのテンプレート実引数がテンプレート仮引数に依存する非型の実引数を含んでいます。
E0520845	［メッセージ］	This partial specialization would have been used to instantiate *名前*.
	［説明］	この部分特別化テンプレートはプライマリテンプレート " *名前*" を実体化しようとしています。
E0520846	［メッセージ］	This partial specialization would have made the instantiation of *名前* ambiguous.
	［説明］	この部分特別化テンプレートは " *名前*" の実体化があいまいになります。
E0520847	［メッセージ］	Expression must have integral or enum type.
	［説明］	式は整数型か列挙型を持つ必要があります。
E0520848	［メッセージ］	Expression must have arithmetic or enum type.
	［説明］	式は算術型か列挙型を持つ必要があります。
E0520849	［メッセージ］	Expression must have arithmetic, enum, or pointer type.
	［説明］	式は算術型か、列挙型、またはポインタ型を持つ必要があります。
E0520850	［メッセージ］	Type of cast must be integral or enum.
	［説明］	キャストの型は整数型か列挙型である必要があります。
E0520851	［メッセージ］	Type of cast must be arithmetic, enum, or pointer.
	［説明］	キャストの型は算術型か、列挙型、またはポインタ型である必要があります。
E0520852	［メッセージ］	Expression must be a pointer to a complete object type.
	［説明］	式は完全型のオブジェクト型へのポインタである必要があります。
E0520854	［メッセージ］	A partial specialization nontype argument must be the name of a nontype parameter or a constant.
	［説明］	一時的な特殊化の型のない引数は型のない引数の名前か低数値でなければなりません。
E0520855	［メッセージ］	Return type is not identical to return type *型* of overridden virtual function *名前*.
	［説明］	関数のリターン型がオーバーライドされた仮想関数 " *名前*" のリターン型 " *型*" と同一でありません。
E0520857	［メッセージ］	A partial specialization of a class template must be declared in the namespace of which it is a member.
	［説明］	部分特別化テンプレートはそのメンバを含む namespace の中で宣言しなければなりません。
E0520858	［メッセージ］	*名前* is a pure virtual function.
	［説明］	" *名前*" は純粋仮想関数です。

E0520859	［メッセージ］	Pure virtual *名前* has no overrider.
	［説明］	純粋仮想関数 " *名前* " はオーバーライドされません。
E0520861	［メッセージ］	Invalid character in input line.
	［説明］	入力行に不正な文字があります。
E0520862	［メッセージ］	Function returns incomplete type " *型名* ".
	［説明］	関数は不完全型 " *型名* " を返します。
E0520864	［メッセージ］	*名前* is not a template.
	［説明］	" *名前* " はテンプレートでありません。
E0520865	［メッセージ］	A friend declaration may not declare a partial specialization.
	［説明］	部分特別化テンプレートはフレンド宣言内で指定できません。
E0520868	［メッセージ］	Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator).
	［説明］	2 つのテンプレート実引数リストの最後に指定する ">>" は間に空白が必要です。
E0520870	［メッセージ］	Invalid multibyte character sequence.
E0520871	［メッセージ］	Template instantiation resulted in unexpected function type of *型 1* (the meaning of a name may have changed since the template declaration -- the type of the template is *型 2*).
	［説明］	" *型 2* " を持つテンプレートの実体化の結果、期待されない型 " *型 1* " の関数が作られました。
E0520872	［メッセージ］	Ambiguous guiding declaration -- more than one function template *名前* matches type *型*.
	［説明］	テンプレート関数があいまいです。
E0520873	［メッセージ］	Non-integral operation not allowed in nontype template argument.
	［説明］	非型のテンプレート実引数に非整数型の演算は許されません。
E0520875	［メッセージ］	Embedded C++ does not support templates.
	［説明］	Embedded C++ 仕様はテンプレート機能をサポートしません。
E0520876	［メッセージ］	Embedded C++ does not support exception handling.
	［説明］	Embedded C++ 仕様は例外処理機能をサポートしません。
E0520877	［メッセージ］	Embedded C++ does not support namespaces.
	［説明］	Embedded C++ 仕様は namespace 機能をサポートしません。
E0520878	［メッセージ］	Embedded C++ does not support run-time type information.
	［説明］	Embedded C++ 仕様はランタイム型情報機能をサポートしません。
E0520879	［メッセージ］	Embedded C++ does not support the new cast syntax.
	［説明］	Embedded C++ 仕様は新形式のキャスト機能をサポートしません。
E0520880	［メッセージ］	Embedded C++ does not support using-declarations.
	［説明］	Embedded C++ 仕様は using 宣言機能をサポートしません。
E0520881	［メッセージ］	Embedded C++ does not support \"mutable\".
	［説明］	Embedded C++ 仕様は mutable 機能をサポートしません。

E0520882	［メッセージ］	Embedded C++ does not support multiple or virtual inheritance.
	［説明］	Embedded C++ 仕様は多重継承／仮想継承機能をサポートしません。
E0520885	［メッセージ］	*型 1* cannot be used to designate constructor for *型 2*.
	［説明］	" *型 1*" はコンストラクタの " *型 2*" で使用することはできません。
E0520886	［メッセージ］	Invalid suffix on integral constant.
	［説明］	整数定数に不正な接尾子（サフィックス）があります。
E0520890	［メッセージ］	Variable length array with unspecified bound is not allowed.
	［説明］	可変長の配列で整列を指定しないのは許されていません。
E0520891	［メッセージ］	An explicit template argument list is not allowed on this declaration.
	［説明］	この宣言内では明示的なテンプレート実引数は許されません。
E0520892	［メッセージ］	An entity with linkage cannot have a type involving a variable length array.
	［説明］	リンケージを伴った実体は可変長配列を持てません。
E0520893	［メッセージ］	A variable length array cannot have static storage duration.
	［説明］	可変長配列は静的な記憶域期間を持てません。
E0520894	［メッセージ］	Entity-kind " *名前* " is not a template
	［説明］	" *名前* " はテンプレートではありません。
E0520896	［メッセージ］	Expected a template argument.
	［説明］	テンプレートの実引数が期待されます。
E0520898	［メッセージ］	Nonmember operator requires a parameter with class or enum type.
	［説明］	非メンバ演算子関数にはクラスまたは列挙型の仮引数が要求されます。
E0520901	［メッセージ］	Qualifier of destructor name *型 1* does not match type *型 2*.
	［説明］	" *型 1*" のデストラクタの限定名が " *型 2*" に一致しません。
E0520915	［メッセージ］	A segment name has already been specified.
	［説明］	セグメント名はすでに指定されています。
E0520916	［メッセージ］	Cannot convert pointer to member of derived class *型1* to pointer to member of base class *型 2* -- base class is virtual.
	［説明］	派生クラス " *型 1*" のメンバへのポインタ型を仮想基底クラス " *型 2*" のメンバへのポインタ型に変換できません。
E0520928	［メッセージ］	Incorrect use of va_start.
	［説明］	va_start の使用法が間違っています。
E0520929	［メッセージ］	Incorrect use of va_arg.
	［説明］	va_arg の使用法が間違っています。
E0520930	［メッセージ］	Incorrect use of va_end.
	［説明］	va_end の使用法が間違っています。
E0520934	［メッセージ］	A member with reference type is not allowed in a union.
	［説明］	参照型は共用体のメンバにできません。

E0520935	［メッセージ］	typedef may not be specified here.
	［説明］	"typedef" はここでは指定できません。
E0520937	［メッセージ］	A class or namespace qualified name is required.
	［説明］	クラスまたは namespace の限定名が要求されます。
E0520938	［メッセージ］	Return type "int" omitted in declaration of function "main".
	［説明］	返却型 "int" が関数 "main" の宣言で省略されました。
E0520939	［メッセージ］	Pointer-to-member representation *xxx* is too restrictive for *xxx*.
	［説明］	メンバへのポインタの宣言が正しくありません。
E0520940	［メッセージ］	Missing return statement at end of non-void *type* "*symbol*".
E0520946	［メッセージ］	Name following "template" must be a template.
	［説明］	"template" に続く名前はメンバテンプレートでなければなりません。
E0520948	［メッセージ］	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
	［説明］	非標準形式のローカルクラスのフレンド宣言です。クラスの定義内に前方宣言がありません。
E0520951	［メッセージ］	Return type of function "main" must be "int".
	［説明］	main 関数の返却型は "int" である必要があります。
E0520952	［メッセージ］	A nontype template parameter may not have class type.
	［説明］	テンプレート仮引数にクラス型名は指定できません。
E0520953	［メッセージ］	A default template argument cannot be specified on the declaration of a member of a class template outside of its class.
	［説明］	クラステンプレートのメンバ宣言にデフォルトのテンプレート実引数を指定できません。
E0520954	［メッセージ］	A return statement is not allowed in a handler of a function try block of a constructor.
	［説明］	コンストラクタの try ブロックのハンドラ内にリターン文は許されません。
E0520955	［メッセージ］	Ordinary and extended designators cannot be combined in an initializer designation.
	［説明］	指示子が正しくありません。
E0520956	［メッセージ］	The second subscript must not be smaller than the first.
	［説明］	2 つ目の添え字は最初のものより小さくてはいけません。
E0520960	［メッセージ］	Type used as constructor name does not match type *型*.
	［説明］	コンストラクタ名として使用された型が "*型*" と一致しません。
E0520961	［メッセージ］	Use of a type with no linkage to declare a variable with linkage.
	［説明］	リンケージを持たない型がリンケージを持つ変数宣言に使用されました。
E0520962	［メッセージ］	Use of a type with no linkage to declare a function.
	［説明］	リンケージを持たない型が関数に使用されました。
E0520963	［メッセージ］	Return type may not be specified on a constructor.
	［説明］	コンストラクタにリターン型を指定できません。

E0520964	［メッセージ］	Return type may not be specified on a destructor.
	［説明］	デストラクタにリターン型を指定できません。
E0520965	［メッセージ］	Incorrectly formed universal character name.
	［説明］	間違ったユニバーサル・キャラクタ名です。
E0520966	［メッセージ］	Universal character name specifies an invalid character.
	［説明］	ユニバーサル・キャラクタ名が不正なキャラクタを指定されました。
E0520967	［メッセージ］	A universal character name cannot designate a character in the basic character set.
	［説明］	ユニバーサル・キャラクタ名は基本的なキャラクタ・セットの文字を指定できません。
E0520968	［メッセージ］	This universal character is not allowed in an identifier.
	［説明］	このユニバーサル・キャラクタは識別子として許されていません。
E0520969	［メッセージ］	The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros.
	［説明］	識別子 __VA_ARGS__ は可変個引数マクロのリストのリプレースのみ使用できます。
E0520971	［メッセージ］	Array range designators cannot be applied to dynamic initializers.
	［説明］	配列範囲名は動的初期化子に適用できません。
E0520972	［メッセージ］	Property name cannot appear here.
	［説明］	プロパティ名はここでは使用できません。
E0520975	［メッセージ］	A variable-length array type is not allowed.
	［説明］	可変長配列は許されていません。
E0520976	［メッセージ］	A compound literal is not allowed in an integral constant expression.
	［説明］	複合リテラルは整数定数式に使用できません。
E0520977	［メッセージ］	A compound literal of type "type" is not allowed.
	［説明］	複合リテラル型 "type" は許されていません。
E0520978	［メッセージ］	A template friend declaration cannot be declared in a local class.
	［説明］	テンプレートのフレンド関数は局所クラスで宣言できません。
E0520979	［メッセージ］	Ambiguous "?" operation: second operand of type *型 1* can be converted to third operand type *型 2*, and vice versa.
	［説明］	三項演算子 "?:" の第 2 式の " *型 1*" と第 3 式の " *型 2*" が互いに変換可能な型であいまいです。
E0520980	［メッセージ］	Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type.
	［説明］	オブジェクトを呼び出していますが operator() 関数または関数へのポインタ型変換関数が定義されていません。
E0520982	［メッセージ］	There is more than one way an object of type "type" can be called for the argument list:
	［説明］	実引数リストから呼ぶことができる " *型*" のオブジェクトが 2 つ以上あります。
E0520983	［メッセージ］	typedef name has already been declared (with similar type).
	［説明］	typedef 名はすでに（同じ型で）宣言されています。

E0520985	［メッセージ］	Storage class "mutable" is not allowed for anonymous unions.
	［説明］	mutable を無名共用体に指定することはできません。
E0520987	［メッセージ］	Abstract class type *型* is not allowed as catch type:
	［説明］	抽象クラスを catch で受けることはできません。
E0520988	［メッセージ］	A qualified function type cannot be used to declare a nonmember function or a static member function.
	［説明］	修飾付き関数型を非メンバ関数や static メンバ関数の宣言に使用することはできません。
E0520989	［メッセージ］	A qualified function type cannot be used to declare a parameter.
	［説明］	修飾された関数型は引数宣言に使用できません。
E0520990	［メッセージ］	Cannot create a pointer or reference to qualified function type.
	［説明］	修飾付き関数型へのポインタ型や参照型を作ることはできません。
E0520992	［メッセージ］	Invalid macro definition:
	［説明］	不正なマクロ定義です。
E0520993	［メッセージ］	Subtraction of pointer types " *型名 1*" and " *型名 2*" is nonstandard.
	［説明］	" *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
E0520994	［メッセージ］	An empty template parameter list is not allowed in a template template parameter declaration.
	［説明］	空テンプレートパラメータを持つテンプレートをテンプレートパラメータに指定することはできません。
E0520995	［メッセージ］	Expected "class".
	［説明］	テンプレートパラメータに指定するクラステンプレートはクラスを必要とします。
E0520996	［メッセージ］	The "class" keyword must be used when declaring a template template parameter.
	［説明］	テンプレートパラメータに指定するクラステンプレートは構造体ではいけません。
E0520998	［メッセージ］	A qualified name is not allowed for a friend declaration that is a function definition.
	［説明］	friend 指定付き関数定義において、名前空間の名前付き関数名を指定することはできません。
E0520999	［メッセージ］	*symbol1* is not compatible with "*symbol2*".
	［説明］	*symbol1* は "*symbol2*" と適合しません。
E0521001	［メッセージ］	Class member designated by a using-declaration must be visible in a direct base class.
	［説明］	クラスメンバの using 指定は参照可能な直接基底クラスでなければなりません。
E0521006	［メッセージ］	A template template parameter cannot have the same name as one of its template parameters.
	［説明］	テンプレートパラメータに指定するクラステンプレート名が、それ自身のテンプレートパラメータ名と同じです。
E0521007	［メッセージ］	Recursive instantiation of default argument.
	［説明］	テンプレート関数のデフォルト引数のインスタンスが再帰的に生成されます。
E0521009	［メッセージ］	*symbol* is not an entity that can be defined.
	［説明］	*symbol* は定義できる実体ではありません。

E0521010	［メッセージ］	Destructor name must be qualified.
	［説明］	不正なデストラクタ名です。
E0521011	［メッセージ］	Friend class name may not be introduced with "typename".
	［説明］	フレンドクラスの名前を “typename” に続けて記述してはいけません。
E0521012	［メッセージ］	A using-declaration may not name a constructor or destructor.
	［説明］	using 宣言でコンストラクタまたはデストラクタを指定してはいけません。
E0521013	［メッセージ］	A qualified friend template declaration must refer to a specific previously declared template.
	［説明］	限定フレンドテンプレートは参照前に定義しておく必要があります。
E0521014	［メッセージ］	Invalid specifier in class template declaration.
	［説明］	不正な指定子がクラステンプレート宣言に含まれています。
E0521015	［メッセージ］	Argument is incompatible with formal parameter.
	［説明］	実引数は仮引数と適合しません。
E0521017	［メッセージ］	Loop in sequence of "operator->" functions starting at class *xxx*.
	［説明］	operator-> が正しくありません。
E0521018	［メッセージ］	*xxx* has no member class *xxx*.
	［説明］	クラスにないメンバを使っています。
E0521019	［メッセージ］	The global scope has no class named *xxx*.
	［説明］	クラス内の名前にファイルスコープ演算子を使っています。
E0521020	［メッセージ］	Recursive instantiation of template default argument.
	［説明］	テンプレートのデフォルト引数で再帰的にインスタンスを生成します。
E0521021	［メッセージ］	Access declarations and using-declarations cannot appear in unions.
	［説明］	union で using 指定は使えません。
E0521022	［メッセージ］	*xxx* is not a class member.
	［説明］	クラスのメンバではありません。
E0521023	［メッセージ］	Nonstandard member constant declaration is not allowed.
	［説明］	標準でない const メンバ宣言は許されていません。
E0521029	［メッセージ］	Type containing an unknown-size array is not allowed.
	［説明］	サイズの不明な配列を持つ型は許されていません。
E0521030	［メッセージ］	A variable with static storage duration cannot be defined within an inline function.
	［説明］	静的変数は inline 関数で定義できません。
E0521031	［メッセージ］	An entity with internal linkage cannot be referenced within an inline function with external linkage.
	［説明］	内部リンケージを持つ識別子は外部リンケージを持つインライン関数内で参照することはできません。
E0521032	［メッセージ］	Argument type %t does not match this type-generic function macro.
	［説明］	引数の型がジェネリック関数生成マクロの型に合いません。

E0521034	［メッセージ］	Friend declaration cannot add default arguments to previous declaration.
	［説明］	フレンド関数が宣言された場合、フレンド関数の定義にデフォルト引数をいれることはできません。
E0521035	［メッセージ］	*xxx* cannot be declared in this scope.
	［説明］	このスコープではテンプレートを宣言することができません。
E0521036	［メッセージ］	The reserved identifier *シンボル* may only be used inside a function.
	［説明］	予約語 " *シンボル* " は関数の中でのみ使用できます。
E0521037	［メッセージ］	This universal character cannot begin an identifierl.
	［説明］	このユニバーサル・キャラクタは識別子の先頭に使用できません。
E0521038	［メッセージ］	Expected a string literal.
	［説明］	文字列リテラルがありません。
E0521039	［メッセージ］	Unrecognized STDC pragma.
	［説明］	認識されない STDC pragma です。
E0521040	［メッセージ］	Expected "ON", "OFF", or "DEFAULT".
	［説明］	"ON"、"OFF" または "DEFAULT" がありません。
E0521041	［メッセージ］	A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.
	［説明］	STDC pragma はグローバルスコープ内の宣言の間か、何かの文またはブロックスコープの宣言の前にのみ置くことががができます。
E0521042	［メッセージ］	Incorrect use of va_copy.
	［説明］	va_copy の使用法が間違っています。
E0521043	［メッセージ］	*xxx* can only be used with floating-point types.
	［説明］	*xxx* は浮動小数点型にのみ使用できます。
E0521044	［メッセージ］	Complex type is not allowed.
	［説明］	複素数型は許されていません。
E0521045	［メッセージ］	Invalid designator kind.
	［説明］	指示子の種類が不正です。
E0521047	［メッセージ］	Complex floating-point operation result is out of range.
	［説明］	複素数型浮動小数点演算の結果が表現可能な値の範囲を越えました。
E0521048	［メッセージ］	Conversion between real and imaginary yields zero.
	［説明］	実数と虚数の相互変換後の値が 0 になりました。
E0521049	［メッセージ］	An initializer cannot be specified for a flexible array member.
	［説明］	可変長配列メンバに初期化子を指定することはできません。
E0521051	［メッセージ］	Standard requires that "symbol" be given a type by a subsequent declaration ("int" assumed).
	［説明］	symbol の型は直後の宣言により与えられなければなりません（"int" に仮定されます）。

E0521052	［メッセージ］	A definition is required for inline "symbol".
	［説明］	インライン関数 "symbol" には定義が必要です。
E0521054	［メッセージ］	A floating-point type must be included in the type specifier for a _Complex or _Imaginary type.
	［説明］	浮動小数点型は複素数または虚数型の指定子に含まれてなければいけません。
E0521055	［メッセージ］	Types cannot be declared in anonymous unions.
	［説明］	型を無名共用体内で宣言することはできません。
E0521056	［メッセージ］	Returning pointer to local variable.
	［説明］	ローカル変数へのポインタを返しています。
E0521057	［メッセージ］	Returning pointer to local temporary.
	［説明］	ローカルな領域へのポインタを返しています。
E0521061	［メッセージ］	Declaration of "symbol" is incompatible with a declaration in another translation unit.
	［説明］	“ シンボル名 ” の宣言はもう一つの翻訳単位内の宣言と互換性がありません。
E0521062	［メッセージ］	The other declaration is %p.
	［説明］	別の宣言があります。
E0521065	［メッセージ］	A field declaration cannot have a type involving a variable length array.
	［説明］	フィールドの宣言は可変長の配列を含む型を持てません。
E0521066	［メッセージ］	Declaration of " *シンボル名* " had a different meaning during compilation of *ファイル名* .
	［説明］	" *シンボル名* " の宣言は *ファイル名* のコンパイル中は異なる意味を持ちます。
E0521067	［メッセージ］	Eexpected "template".
	［説明］	"template" がありません。
E0521072	［メッセージ］	A declaration cannot have a label.
	［説明］	宣言はラベルを持てません。
E0521075	［メッセージ］	*シンボル名* already defined during compilation of *xxx*.
	［説明］	" *シンボル名* " は *xxx* のコンパイル時にすでに定義されています。
E0521076	［メッセージ］	*シンボル名* already defined in another translation unit.
	［説明］	" *シンボル名* " は他のコンパイル単位ですでに定義されています。
E0521081	［メッセージ］	A field with the same name as its class cannot be declared in a class with a user-declared constructor.
	［説明］	クラス名と同じ名前のメンバを宣言することはできません。
E0521086	［メッセージ］	The object has cv-qualifiers that are not compatible with the member "*symbol*".
	［説明］	オブジェクトはメンバ "*symbol*" と合致しない const または volatile 修飾子を持ちます。
E0521087	［メッセージ］	No instance of *xxx* matches the argument list and object (the object has cv-qualifiers that prevent a match).
	［説明］	“ *クラス名* ” のインスタンスは引数リストとオブジェクトと合致しません（オブジェクトの持つ CV 修飾子が合致を抑止しています）。

E0521088	［メッセージ］	An attribute specifies a mode incompatible with *xxx*.
	［説明］	幅が指定された型がありません。
E0521089	［メッセージ］	There is no type with the width specified.
	［説明］	指定された幅を持つ型がありません。
E0521139	［メッセージ］	The "template" keyword used for syntactic disambiguation may only be used within a template.
	［説明］	キーワード ”template” を構文上のあいまいさを解消するのに使用できるのは template 内のみです。
E0521144	［メッセージ］	Storage class must be auto or register.
	［説明］	記憶クラスは auto または register でなければいけません。
E0521146	［メッセージ］	*xxx* is not a base class member.
	［説明］	基底クラスのメンバではありません。
E0521158	［メッセージ］	void return type cannot be qualified.
	［説明］	void の返却型は修飾できません。
E0521161	［メッセージ］	A member template corresponding to *xxx* is declared as a template of a different kind in another translation unit.
	［説明］	テンプレート宣言が他コンパイル単位と異なっています。
E0521163	［メッセージ］	va_start should only appear in a function with an ellipsis parameter.
	［説明］	va_start が使用されるのは省略記号を引数とする関数のみです。
E0521201	［メッセージ］	typedef *xxx* may not be used in an elaborated type specifier.
	［説明］	typedef *xxx* は記述型指定子では使用できません。
E0521203	［メッセージ］	Parameter *引数名* may not be redeclared in a catch clause of function try block.
	［説明］	" *引数名* " を try ブロックの catch 句の中で再宣言してはいけません。
E0521204	［メッセージ］	The initial explicit specialization of *xxx* must be declared in the namespace containing the template.
	［説明］	シンボルに対する最初の明示的な特殊化はテンプレートを含む名前空間の中に宣言されなければいけません。
E0521206	［メッセージ］	"template" must be followed by an identifier.
	［説明］	"template” の後には識別子が必要です。
E0521227	［メッセージ］	Transfer of control into a statement expression is not allowed.
	［説明］	文式中への制御の移動は許されていません。
E0521229	［メッセージ］	This statement is not allowed inside of a statement expression.
	［説明］	この文は文式の中では許されていません。
E0521230	［メッセージ］	Anon-POD class definition is not allowed inside of a statement expression.
	［説明］	非 POD クラスは式文内に定義できません。
E0521254	［メッセージ］	Integer overflow in internal computation due to size or complexity of " *型* ".
	［説明］	" *型* " のサイズまたは複雑さのため内部計算時にオーバフローを起こしました。
E0521255	［メッセージ］	Integer overflow in internal computation.

E0521273	［メッセージ］	Alignment-of operator applied to incomplete type.
	［説明］	オペレータのアライメントが不完全な型に対して適用されました。
E0521280	［メッセージ］	Conversion from inaccessible base class *xxx* is not allowed.s
	［説明］	派生クラスにプライベートで継承された基底クラス型のポインタを継承クラス型のポインタへ変換することはできません。
E0521282	［メッセージ］	String literals with different character kinds cannot be concatenated.
	［説明］	種類の異なる文字を持つ文字列リテラルは連結できません。
E0521291	［メッセージ］	A non-POD class type cannot be fetched by va_arg.
	［説明］	非 POD 型のクラスは va_arg によって取得することができません。
E0521292	［メッセージ］	The 'u' or 'U' suffix must appear before the 'l' or 'L' suffix in a fixed-point literal.
	［説明］	型接尾語 'u' または 'U' は固定小数点リテラル中の型接尾語 'l' または 'L' の前にある必要があります。
E0521295	［メッセージ］	Fixed-point constant is out of range.
	［説明］	固定小数点定数が範囲を越えています。
E0521303	［メッセージ］	Expression must have integral, enum, or fixed-point type.
	［説明］	式は整数か列挙対または固定小数点型である必要があります。
E0521304	［メッセージ］	Expression must have integral or fixed-point type.
	［説明］	式は整数か固定小数点型である必要があります。
E0521311	［メッセージ］	Fixed-point types have no classification.
	［説明］	浮動小数点型の区分がありません。
E0521312	［メッセージ］	A template parameter may not have fixed-point type.
	［説明］	テンプレート引数には固定小数点型を指定できません。
E0521313	［メッセージ］	Hexadecimal floating-point constants are not allowed.
	［説明］	16 進の浮動小数点定数は許されていません。
E0521315	［メッセージ］	Floating-point value does not fit in required fixed-point type.
	［説明］	浮動小数値が要求された浮動小数点型に適していません。
E0521317	［メッセージ］	Fixed-point conversion resulted in a change of sign.
	［説明］	負の整数値を固定小数点型へ変換した結果、正の値になりました。
E0521318	［メッセージ］	Integer value does not fit in required fixed-point type.
	［説明］	整数値が要求された固定小数点型に適していません。
E0521319	［メッセージ］	Fixed-point operation result is out of range.
	［説明］	固定小数点の処理が範囲を越えました。
E0521320	［メッセージ］	Multiple named address spaces.
	［説明］	同一の名前アドレス空間が複数存在します。
E0521321	［メッセージ］	Variable with automatic storage duration cannot be stored in a named address space.
	［説明］	局所的なスコープを持つ変数は名前付きアドレス空間に保持することはできません。

E0521322	［メッセージ］	Type cannot be qualified with named address space.
	［説明］	名前付きアドレス空間によって型を識別することはできません。
E0521323	［メッセージ］	Function type cannot be qualified with named address space.
	［説明］	名前付きアドレス空間によって関数型を識別することはできません。
E0521324	［メッセージ］	Field type cannot be qualified with named address space.
	［説明］	フィールド型は名前付き空間によって識別することはできません。
E0521325	［メッセージ］	Fixed-point value does not fit in required floating-point type.
	［説明］	固定小数点値は要求された浮動小数点型に収まりません。
E0521326	［メッセージ］	Fixed-point value does not fit in required integer type.
	［説明］	固定小数点値は要求された整数型に収まりません。
E0521327	［メッセージ］	Value does not fit in required fixed-point type.
	［説明］	値は要求された固定小数点値に収まりません。
E0521344	［メッセージ］	A named address space qualifier is not allowed here.
	［説明］	名前付きアドレス空間識別子はここで使用できません。
E0521345	［メッセージ］	An empty initializer is invalid for an array with unspecified bound.
	［説明］	バウンドが指定されていない配列の初期化への空の初期化子は不正です。
E0521348	［メッセージ］	Declaration hides "*symbol*".
	［説明］	宣言は "*symbol*" を隠します。
E0521349	［メッセージ］	A parameter cannot be allocated in a named address space.
	［説明］	引数は名前付きアドレス空間に配置できません。
E0521350	［メッセージ］	Invalid suffix on fixed-point or floating-point constant.
	［説明］	不正な接尾辞が固定または浮動小数点定数についています。
E0521351	［メッセージ］	A register variable cannot be allocated in a named address space.
	［説明］	レジスタ変数は名前付きアドレス空間に配置できません。
E0521352	［メッセージ］	Expected "SAT" or "DEFAULT".
	［説明］	"SAT" か "DEFAULT" がありません。
E0521355	［メッセージ］	A function return type cannot be qualified with a named address space.
	［説明］	関数の戻り値を名前付きアドレス空間で修飾することはできません。
E0521365	［メッセージ］	Named-register variables cannot have void type.
	［説明］	名前付きレジスタ変数は void 型にできません。
E0521372	［メッセージ］	Nonstandard qualified name in global scope declaration.
	［説明］	非標準形式で修飾した名前がグローバルなスコープに宣言されています。
E0521380	［メッセージ］	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
	［説明］	仮想関数の定義がありません。また、無名空間のメンバであるため、それ以外の場所で定義することができません。

E0521381	［メッセージ］	Carriage return character in source line outside of comment or character/string literal.
	［説明］	改行文字がコメントまたは文字列リテラル以外のところにあります。
E0521382	［メッセージ］	Expression must have fixed-point type.
	［説明］	式は固定小数点型である必要があります。
E0521398	［メッセージ］	Invalid member for anonymous member class -- class *xxx* has a disallowed member function.
	［説明］	無名のメンバクラスに対して不正なメンバ関数を宣言しています。
E0521403	［メッセージ］	A variable-length array is not allowed in a function return type.
	［説明］	可変長の配列は関数の返却値にできません。
E0521404	［メッセージ］	Variable-length array type is not allowed in pointer to member of type "*type*".
	［説明］	可変長配列型は "*type*" 型のメンバへのポインタではできません。
E0521405	［メッセージ］	The result of a statement expression cannot have a type involving a variable-length array.
	［説明］	文式の結果は可変長配列に関する型を持てません。
E0521420	［メッセージ］	Some enumerator values cannot be represented by the integral type underlying the enum type.
	［説明］	いくつかの列挙子はその列挙型の潜在的な整数型で表現できません。
E0521424	［メッセージ］	Second operand of offsetof must be a field.
	［説明］	2 つ目の offsetof のオペランドはフィールドである必要があります。
E0521425	［メッセージ］	Second operand of offsetof may not be a bit field.
	［説明］	2 つ目の offsetof のオペランドはビットフィールドではなりません。
E0521436	［メッセージ］	*xxx* is only allowed in C.
	［説明］	*xxx* は C 言語のみで許されています。
E0521437	［メッセージ］	__ptr32 and __ptr64 must follow a "*".
	［説明］	__ptr32 と __ptr64 は "*" に続く必要があります。
E0521441	［メッセージ］	Complex integral types are not supported.
	［説明］	複素数型はサポートされていません。
E0521442	［メッセージ］	__real and __imag can only be applied to complex values.
	［説明］	__real と __imag は複素数値にのみ使用できます。
E0521445	［メッセージ］	Invalid redefinition of "*symbol*".
	［説明］	*symbol* の不正な再定義です。
E0521534	［メッセージ］	Duplicate function modifier.
	［説明］	関数修飾子が重複しています。
E0521535	［メッセージ］	Invalid character for char16_t literal.
	［説明］	char16_t リテラルが不正です。
E0521536	［メッセージ］	__LPREFIX cannot be applied to char16_t or char32_t literals.
	［説明］	__LPREFIX は char16_t か char32_t のリテラルに指定できません。

E0521537	［メッセージ］	Unrecognized calling convention *xxx* must be one of:
	［説明］	許されていないコーリングコンベンション *xxx* です。以下の一つである必要があります :
E0521539	［メッセージ］	Option "--uliterals" can be used only when compiling C.
	［説明］	--uliterals オプションは C のコンパイル時にのみ使用できます。
E0521542	［メッセージ］	Some enumerator constants cannot be represented by "type".
	［説明］	いくつかの列挙子定数は "type" で表現できません。
E0521543	［メッセージ］	*xxx* not allowed in current mode.
	［説明］	*xxx* は現在のモードで許されていません。
E0521557	［メッセージ］	Alias creates cycle of aliased entities.
	［説明］	エイリアスがエイリアスされた実体に循環しています。
E0521558	［メッセージ］	Subscript must be constant.
	［説明］	添え字は定数である必要があります。
E0521574	［メッセージ］	Static assertion failed with *xxx*.
	［説明］	静的なアサーションは失敗しました。
E0521576	［メッセージ］	Field name resolves to more than one offset -- see "*symbol1*" and "*symbol2*".
	［説明］	フィールド名は複数のオフセットがあります -- "*symbol1*" と "*symbol2*" を見てください。
E0521577	［メッセージ］	*xxx* is not a field name.
	［説明］	*xxx* はフィールド名ではありません。
E0521578	［メッセージ］	case label value has already appeared in this switch *行番号*
	［説明］	case ラベル値はすでにこの switch の行*行番号*で現れています。
E0521582	［メッセージ］	The option to list macro definitions may not be specified when compiling more than one translation unit.
	［説明］	マクロ定義を記載するオプションは複数ファイルコンパイル時に指定できません。
E0521583	［メッセージ］	Unexpected parenthesis after declaration of "symbol" (malformed parameter list or invalid initializer?).
	［説明］	symbol の宣言の後に予期しないかっこがあります（正常でない引数リストか不正な初期化子の可能性があります）。
E0521584	［メッセージ］	Parentheses around a string initializer are nonstandard.
	［説明］	文字列リテラルをかっこで囲むことは標準ではありません。
E0521586	［メッセージ］	A variable declared with an auto type specifier cannot appear in its own initializer.
	［説明］	auto 型を指定され宣言された変数はそれ自身の初期化子には指定できません。
E0521587	［メッセージ］	Cannot deduce "auto" type.
	［説明］	auto 型を推測できません。
E0521588	［メッセージ］	Initialization with "{...}" is not allowed for "auto" type.
	［説明］	{...} を伴った初期化は "auto" 型に指定できません。

E0521589	［メッセージ］	auto type cannot appear in top-level array type.
	［説明］	auto 型はトップ・レベルの配列型に指定できません。
E0521590	［メッセージ］	auto type cannot appear in top-level function type.
	［説明］	auto 型はトップ・レベルの関数型には指定できません。
E0521593	［メッセージ］	Cannot deduce "auto" type (initializer required).
	［説明］	auto 型を推測できません（初期化子が必要です）。
E0521596	［メッセージ］	Invalid use of a type qualifier.
	［説明］	不正な型修飾子の使用です。
E0521597	［メッセージ］	A union cannot be abstract or sealed.
	［説明］	共用体に abstract または sealed は指定できません。
E0521598	［メッセージ］	auto is not allowed here.
	［説明］	auto はここでは許されていません。
E0521602	［メッセージ］	struct/union variable "*variable*" with a member of incomplete type cannot be placed into the section.
	［説明］	不完全型をメンバにもつ構造体 / 共用体変数 " *変数名* " はセクションを指定できません 。
E0521603	［メッセージ］	Variable of incomplete type "*variable*" cannot be placed into the section.
	［説明］	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］	Illegal section attribute.
	［説明］	#pragma section で指定された再配置属性が不正です。
E0521605	［メッセージ］	Illegal #pragma *character* string syntax.
	［説明］	#pragma *文字列* の文法が不正です。
E0521606	［メッセージ］	"*function*" has already been placed into another section.
	［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 is not allowed outside of function.
	［説明］	#pragma asm は関数の外に記述できません。
E0521609	［メッセージ］	The #pragma endasm for this #pragma asm is missing.
	［説明］	#pragma asm が #pragma endasm によって閉じられていません。
E0521610	［メッセージ］	The #pragma asm for this #pragma endasm is missing.
	［説明］	#pragma asm なしに #pragma endasm が記述されています。
E0521612	［メッセージ］	Duplicate interrupt hander for "*request*".
	［説明］	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］	Interrupt request name "*request*" not supported.
	［説明］	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］	Duplicate #pragma interrupt for this function.
	［説明］	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]	Duplicate #pragma smart_correct for this function "*function*".
	[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]	*Type* "*symbol*" has already been placed into another section (declared as extern).
	[説明]	*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]	*Type* "*symbol*" has already been placed into another section.
	[説明]	*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]	*Type* "*symbol*" has already been declared with #pragma section.
	[説明]	*種別* " *シンボル名* " はすでにセクションが指定されています。新たにセクションなしで宣言することはできません。
E0521619	[メッセージ]	*Type* "*symbol*" has already been declared without #pragma section.
	[説明]	*種別* " *シンボル名* " はすでにセクションなしで宣言されています。新たにセクション指定することはできません。
E0521620	[メッセージ]	"*function*()" argument overflow. use "*minimum value - maximum value*".
	[説明]	組み込み関数 " *関数名* ()" の引数の値が範囲を越えています。" *最小値～最大値* " が指定できます。
E0521621	[メッセージ]	Cannot write I/O register "*register name*".
	[説明]	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]	Cannot read I/O register "*register name*".
	[説明]	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]	Cannot use *expanded specification*. Device must be specified.
	[説明]	デバイスの指定がない場合*拡張機能指定*の機能は使用できません。
E0521624	[メッセージ]	Second argument for __set_il()must be string literal.
	[説明]	__set_il() の第二引数は文字列定数である必要があります。
E0521625	[メッセージ]	Cannot set interrupt level for "*request*".
	[説明]	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]	*Specification character string* is specified for function "*function name*", previously specified #pragma inline is ignored.
	[説明]	*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]	Function for #pragma smart_correct is same.
	[説明]	#pragma smart_correct に指定された関数が同じです。
E0521628	[メッセージ]	Function for #pragma smart_correct "*function*" is undefined.
	[説明]	#pragma smart_correct に指定された関数 " *関数名* " が定義されていません。
E0521629	[メッセージ]	Could not open symbol file "*file name*".
	[説明]	シンボル・ファイル " *ファイル名* " をオープンすることができません。
E0521630	[メッセージ]	Could not close symbol file "*file name*".
	[説明]	シンボル・ファイル " *ファイル名* " をクローズすることができません。

E0521631	［メッセージ］	Syntax error in symbol file.
	［説明］	シンボル・ファイル中に構文エラーがありました。
E0521632	［メッセージ］	Unrecognized symbol information "*character string*" is ignored.
	［説明］	" *文字列* " は認識できない情報行です。無視します。
E0521633	［メッセージ］	Section name is not specified.
	［説明］	セクション名が指定されていません。
E0521634	［メッセージ］	Unrecognized section name "*section*".
	［説明］	不正なセクション名 " *セクション名* " が指定されています。
E0521635	［メッセージ］	"*variable name*" has already been placed into "*section name*" section in symbol file. The latter is ignored.
	［説明］	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］	"*variable name*" has already been placed into "*section name*" section in symbol file. #pragma is ignored.
	［説明］	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］	Illegal binary digit.
	［説明］	2 進定数が不正です。
E0521638	［メッセージ］	First argument for *special function name*()must be integer constant.
	［説明］	*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］	Function "*function name*" specified as "direct" can not be allocated in text.
	［説明］	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］	Function allocated in text can not be specified #pragma interrupt with "direct".
	［説明］	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 level interrupt not supported.
	［説明］	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］	Cannnot give a name for "*attribute*" section.
	［説明］	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" cannot be specified for plural interrupt.
	［説明］	"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］	Reduced exception handler option of device is available. Address of the handlermaybe overlaps.
	［説明］	デバイスの例外ハンドラ縮小機能が有効になっています。EI レベル・マスカブル割り込みのハンドラアドレスが重複する可能性があります。
E0521645	［メッセージ］	Function "*function name*" has illegal type for interrupt function,must be void(void).
	［説明］	void 型ではない関数 " *関数名* " は割り込み関数にできません。
E0521646	［メッセージ］	Cannot use direct with NO_VECT.
	［説明］	割り込み要求名に NO_VECT を指定した場合、"direct" 指定できません。

E0521647	［メッセージ］	*character string* is not allowed here.
	［説明］	*文字列*はここでは許されていません。
E0521648	［メッセージ］	Cannot call type function "*function name*".
	［説明］	*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］	White space is required between the macro name *xxx* and its replacement text.
	［対処方法］	マクロ名とその置換テキストの間に空白を入れて区切ってください。
E0521650	［メッセージ］	type "symbol name" has already been declared with other #pragma pic/nopic.
	［説明］	*種別* " *シンボル名* " に対して矛盾する #pragma pic/nopic 指定がされています。
E0523005	［メッセージ］	Invalid pragma declaration.
	［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。
E0523006	［メッセージ］	"*symbol name*" has already been specified by other pragma.
	［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］	Pragma may not be specified after definition
	［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］	Invalid kind of pragma is specified to this symbol
	［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。
E0523042	［メッセージ］	Using "*function item*" function at influence the code generation of "SuperH" compiler
	［説明］	SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
E0523047	［メッセージ］	Illegal #pragma interrupt declaration
	［説明］	割り込み関数宣言 #pragma interrupt の仕様に誤りがあります。
E0523049	［メッセージ］	Multiple #pragma for one function
	［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。
E0523057	［メッセージ］	Illegal section specified.
	［説明］	セクションの属性指定文字として、使用不可能な文字列を使用しました。
E0523058	［メッセージ］	Illegal #pragma section syntax
	［説明］	#pragma section の構文が文法に違反しています。
E0523059	［メッセージ］	Cannot change text section.
	［説明］	#pragma section の構文が正しくありません。
E0523061	［メッセージ］	Argument is incompatible with formal parameter of intrinsic function.
	［説明］	実引数は組み込み関数の仮引数と適合しません。
E0523062	［メッセージ］	Return value type does not match the intrinsic function type.
	［説明］	返却値の型が組み込み関数の型と合っていません。
E0523065	［メッセージ］	Cannot assign address constant to initializer for bitfield
	［説明］	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。
E0523066	［メッセージ］	The combination of the option and section specification is inaccurate.
	［説明］	このセクションは現在のオプションでは使用できません。

E0523129	［メッセージ］	" *オプション名* " option is necessary for use of " *機能* ".
	［説明］	この機能を使用するにはオプションを指定する必要があります。
E0532002	［メッセージ］	Exception *exception* has occurred at compile time.
E0544003	［メッセージ］	The size of " *セクション名* " section exceeds the limit.
	［説明］	" *セクション名* " セクションのサイズが上限値を越えています。
E0544240	［メッセージ］	Illegal naming of section " *セクション名* ".
	［説明］	セクションの命名に誤りがあります。用途の異なるセクションに同じ名前が付いています。
E0544854	［メッセージ］	Illegal address was specified with #pragma address.
	［説明］	#pragma address によるアドレスの指定が以下のいずれかに該当しています。 (1) 異なる変数に対して、同一アドレスを指定している。 (2) 異なる変数に対して、変数のアドレスが重なっている。
E0552000	［メッセージ］	No space after mnemonic or directive.
	［説明］	ニーモニック、アセンブル制御命令の直後に空白文字がありません。
	［対処方法］	命令とオペランドの間に、空白文字を記述してください。
E0552001	［メッセージ］	',' is missing.
	［説明］	',' の記述がありません。
	［対処方法］	オペランドの区切りには、カンマを記述してください。
E0552002	［メッセージ］	Characters exist in expression.
	［説明］	命令又は式中に余分な文字があります。
	［対処方法］	式の記述規則を確認してください。
E0552003	［メッセージ］	Size specifier is missing.
	［説明］	サイズ指定子がありません。
	［対処方法］	サイズ指定子を記述してください。
E0552004	［メッセージ］	Invalid operand(s) exist in instruction.
	［説明］	命令に無効なオペランドがあります。
	［対処方法］	命令のオペランドの記述方法を確認して、記述し直してください。
E0552005	［メッセージ］	Operand type is not appropriate.
	［説明］	オペランドの種類が間違っています。
	［対処方法］	オペランドの記述方法を確認して、記述し直してください。
E0552006	［メッセージ］	Size specifier is not appropriate.
	［説明］	サイズ指定子の記述に間違いがあります。
	［対処方法］	サイズ指定子を記述し直してください。
E0552007	［メッセージ］	Operand label is not in the same section.
	［説明］	分岐先が同一セクション内にありません。
	［対処方法］	同一セクション内の分岐先にしか分岐できません。ニーモニックを記述し直してください。

E0552008	［メッセージ］	Illegal displacement value.
	［説明］	ディスプレースメント値が間違っています。
	［対処方法］	サイズ指定子が W のときは、2 の倍数、L のときは、4 の倍数にしてください。
E0552009	［メッセージ］	FPU instruction or FPSW is used.
	［説明］	浮動小数点演算（FPU）命令または FPSW を使用しています。
	［対処方法］	CPU 種別を確認してください。
E0552010	［メッセージ］	ISAV2 instruction or EXTB is used
	［説明］	ISAV2 拡張命令、または EXTB を使用しています。
	［対処方法］	-isa オプション、または環境変数 ISA_RX による RX 命令セットアーキテクチャの選択を確認してください。
E0552011	［メッセージ］	*種別* instruction is used.
	［説明］	*種別*の命令を使用しています。
	［対処方法］	現在のオプション、または環境変数の設定では*種別*の命令を使用できません。設定内容を見直してください。
E0552020	［メッセージ］	Invalid operand(s) exist in debug information
	［説明］	.line 疑似命令のオペランドが不正です。
E0552022	［メッセージ］	Symbol name is missing.
	［説明］	EQU 制御命令行にシンボル名が未記述です。
	［対処方法］	シンボル名を記述してください。
E0552023	［メッセージ］	Illegal directive command is used.
	［説明］	不正な制御命令を記述しています。
	［対処方法］	正しい制御命令に記述し直してください。
E0552024	［メッセージ］	No ';' at the top of comment.
	［説明］	コメント先頭に ";" が記述されていません。
	［対処方法］	コメントの先頭には、セミコロンを記述してください。ニーモニック、またはオペランドの記述に誤りがないか確認してください。
E0552026	［メッセージ］	'CODE' section in big endian is not appropriate.
	［説明］	endian=big 時、絶対属性の CODE セクション開始アドレスに 4 の倍数以外の値を指定しています。
	［対処方法］	絶対属性の CODE セクション開始アドレスには 4 の倍数の値を指定してください。
E0552027	［メッセージ］	Illegal character code.
	［説明］	文字コードが正しくありません。
E0552028	［メッセージ］	Unrecognized character escape sequence.
	［説明］	認識できないエスケープシーケンスがあります。
E0552029	［メッセージ］	Invalid description in #pragma inline_asm function.
	［説明］	アセンブラ埋め込みインライン関数のアセンブリ言語に、使用できない記述があります。

E0552040	［メッセージ］	Include nesting over.
	［説明］	インクルードのネストレベルが深すぎます。
	［対処方法］	インクルードのネスとレベルが 9 以下になるように記述し直してください。
E0552041	［メッセージ］	Can't open include file 'XXXX'.
	［説明］	インクルード・ファイルをオープンできません。
	［対処方法］	インクルード・ファイル名を確認してください。インクルード・ファイルの格納ディレクトリを確認してください。
E0552042	［メッセージ］	Including the include file in itself.
	［説明］	インクルード・ファイル内で、自身をインクルードしています。
	［対処方法］	インクルード・ファイル名を確認して、記述し直してください。
E0552049	［メッセージ］	Invalid reserved word exist in operand.
	［説明］	オペランド中に予約語が記述されています。
	［対処方法］	予約語は、オペランドに記述できません。オペランドを記述し直してください。
E0552050	［メッセージ］	Operand value is not defined.
	［説明］	オペランドの値が未定義です。
	［対処方法］	オペランドには確定値を記述してください。
E0552051	［メッセージ］	'{' is missing.
	［説明］	'{' の記述がありません。
E0552052	［メッセージ］	Addressing mode specifier is not appropriate.
	［説明］	アドレッシングモード指定子の記述に間違いがあります。
	［対処方法］	アドレッシングモード指定子の記述方法を確認してください。
E0552053	［メッセージ］	Reserved word is missing.
	［説明］	予約語の記述がありません。
E0552054	［メッセージ］	']' is missing.
	［説明］	'] ' の記述がありません。
	［対処方法］	'[' に対応する ']' を記述してください。
E0552055	［メッセージ］	Right quote is missing.
	［説明］	右側の引用符がありません。
	［対処方法］	引用符を記述してください。
E0552056	［メッセージ］	The value is not constant.
	［説明］	値がアセンブル時確定値ではありません。
	［対処方法］	アセンブル時に確定するような、式、シンボル名又はラベル名を記述してください。
E0552057	［メッセージ］	Quote is missing.
	［説明］	文字列に対する引用符の記述がありません。
	［対処方法］	文字列は引用符で囲って記述してください。

E0552058	［メッセージ］	Illegal operand is used.
	［説明］	オペランドが間違っています。
	［対処方法］	オペランドの記述方法を確認して、記述し直してください。
E0552059	［メッセージ］	Operand number is not enough.
	［説明］	オペランドが不足しています。
	［対処方法］	オペランドの記述方法を確認して、記述し直してください。
E0552060	［メッセージ］	Too many macro nesting.
	［説明］	マクロのネスティングが多すぎます。
	［対処方法］	マクロのネスティングレベルを 65535 レベル以下にしてください。ソース記述を確認してください。
E0552061	［メッセージ］	Too many macro local label definition.
	［説明］	マクロ内ローカルラベルの定義が多すぎます。
	［対処方法］	マクロ内ローカルラベル数を 1 ファイル内に 65535 個以下にしてください。
E0552062	［メッセージ］	'.MACRO' is missing for '.ENDM'.
	［説明］	.ENDM に対する .MACRO がありません。
	［対処方法］	.ENDM の記述位置を確認してください。
E0552063	［メッセージ］	'.MREPEAT' is missing for '.ENDR'.
	［説明］	.ENDR に対する .MREPEAT がありません。
	［対処方法］	.ENDR の記述位置を確認してください。
E0552064	［メッセージ］	'.MACRO' or '.MREPEAT' is missing for '.EXITM'.
	［説明］	.EXITM に対する .MACRO 又は .MREPEAT がありません。
	［対処方法］	.EXITM の記述位置を確認してください。
E0552065	［メッセージ］	No macro name.
	［説明］	マクロ名がありません。
	［対処方法］	マクロ定義には、マクロ名を記述してください。
E0552066	［メッセージ］	Too many formal parameter.
	［説明］	マクロの仮引数の定義数が多すぎます。
	［対処方法］	マクロの仮引数の数を 80 以下にしてください。
E0552067	［メッセージ］	Illegal macro parameter.
	［説明］	マクロ引数に不正な記述があります。
	［対処方法］	マクロ引数の記述内容を確認してください。
E0552068	［メッセージ］	Source line is too long.
	［説明］	ソース行が長すぎます。
	［対処方法］	ソース行の記述内容を確認してください。

E0552069	［メッセージ］	'.MACRO' is missing for '.LOCAL'.
	［説明］	.LOCAL に対する .MACRO がありません。
	［対処方法］	.LOCAL の記述位置を確認してください。.LOCAL は、マクロブロック内にしか記述できません。
E0552070	［メッセージ］	No '.ENDM' statement.
	［説明］	.ENDM 記述がありません。
	［対処方法］	.ENDM の記述位置を確認してください。.ENDM を記述してください。
E0552071	［メッセージ］	No '.ENDR' statement.
	［説明］	.ENDR 記述がありません。
	［対処方法］	.ENDR の記述位置を確認してください。.ENDR を記述してください。
E0552072	［メッセージ］	')' is missing.
	［説明］	')' の記述がありません。
	［対処方法］	'(' に対応する ')' を記述してください。
E0552073	［メッセージ］	Operand expression is not completed.
	［説明］	オペランド記述に不足があります。
	［対処方法］	オペランドの記述方法を確認して、記述し直してください。
E0552074	［メッセージ］	Syntax error in expression.
	［説明］	式の記述に間違いがあります。
	［対処方法］	式の記述方法を確認して、記述し直してください。
E0552075	［メッセージ］	String value exist in expression.
	［説明］	式中に文字列式が記述されています。
	［対処方法］	式を記述し直してください。
E0552076	［メッセージ］	Division by zero.
	［説明］	除数 0 による除算が行われています。
	［対処方法］	式を記述し直してください。
E0552077	［メッセージ］	No '.END' statement.
	［説明］	.END の記述がありません。
	［対処方法］	ソースプログラムの最後の行に .END を記述してください。
E0552078	［メッセージ］	The specified address overlaps at ' *アドレス値* '.
	［説明］	指定された ' *アドレス値* ' はアドレス割付けが重複しています。C/C++ ソースの場合は ' *アドレス値* ' で複数のシンボルが重複しています。
	［対処方法］	.ORG、.OFFSET の指定内容を見直してください。' *アドレス値* ' に割り当たっているシンボルを確認してください。
E0552080	［メッセージ］	'.IF' is missing for '.ELSE'.
	［説明］	.ELSE に対する .IF がありません。
	［対処方法］	.ELSE の記述位置を確認してください。

E0552081	［メッセージ］	'.IF' is missing for '.ELIF'.
	［説明］	.ELIF に対する .IF がありません。
	［対処方法］	.ELIF の記述位置を確認してください。
E0552082	［メッセージ］	'.IF' is missing for '.ENDIF'.
	［説明］	.ENDIF に対する .IF がありません。
	［対処方法］	.ENDIF の記述位置を確認してください。
E0552083	［メッセージ］	Too many nesting level of condition assemble.
	［説明］	条件アセンブルのネスティングが多すぎます。
	［対処方法］	条件アセンブルの記述を確認してください。
E0552084	［メッセージ］	No '.ENDIF' statement.
	［説明］	ソース・ファイル内に IF 文に対応した ENDIF がありません。
	［対処方法］	ソースの記述を確認してください。
E0552088	［メッセージ］	Can't open '.ASSERT' message file 'XXXX'.
	［説明］	.ASSERT の出力ファイルをオープンできません。
	［対処方法］	ファイル名を確認してください。
E0552089	［メッセージ］	Can't write '.ASSERT' message file 'XXXX'.
	［説明］	.ASSERT の出力ファイルに書き込みできません。
	［対処方法］	ファイルのパーミッションを確認してください。
E0552090	［メッセージ］	Too many temporary label.
	［説明］	テンポラリラベルの個数が多すぎます。
	［対処方法］	テンポラリラベルをラベル名に置き換えて記述してください。
E0552091	［メッセージ］	Temporary label is undefined.
	［説明］	テンポラリラベルが未定義です。
	［対処方法］	テンポラリラベルの定義を行ってください。
E0552100	［メッセージ］	Value is out of range.
	［説明］	値が範囲外です。
	［対処方法］	レジスタなどのビット長に合った値を記述してください。
E0552112	［メッセージ］	Symbol is missing.
	［説明］	シンボルの記述がありません。
	［対処方法］	シンボル名を記述してください。
E0552113	［メッセージ］	Symbol definition is not appropriate.
	［説明］	シンボルの定義に間違いがあります。
	［対処方法］	シンボル定義方法を確認して記述し直してください。
E0552114	［メッセージ］	Symbol has already defined as another type.
	［説明］	シンボルは既に同一名で異なる制御命令で定義されています。
	［対処方法］	シンボル名を変更してください。

E0552115	［メッセージ］	Symbol has already defined as the same type.
	［説明］	シンボルは、すでに定義されています。
	［対処方法］	シンボル名を変更してください。
E0552116	［メッセージ］	Symbol is multiple defined.
	［説明］	シンボルが二重定義です。マクロ名と他の名前が重複しています。
	［対処方法］	シンボル名を変更してください。
E0552117	［メッセージ］	Invalid label definition.
	［説明］	無効なラベル記述をしています。
	［対処方法］	ラベル定義を記述し直してください。
E0552118	［メッセージ］	Invalid symbol definition.
	［説明］	無効なシンボル記述をしています。
	［対処方法］	シンボルの定義を記述し直してください。
E0552119	［メッセージ］	Reserved word is used as label or symbol.
	［説明］	予約語をラベル、またはシンボルに用いています。
	［対処方法］	ラベル、またはシンボル名を記述し直してください。
E0552120	［メッセージ］	Created symbol is too long.
	［説明］	-create_unfilled_area オプションで作成された予約語のラベルが長すぎます。
	［対処方法］	ファイル、セクション名を短くしてください。
E0552130	［メッセージ］	No '.SECTION' statement.
	［説明］	'.SECTION ' の記述がありません。
	［対処方法］	ソースプログラムには、必ず 1 つ以上の .SECTION を記述してください。
E0552131	［メッセージ］	Section type is not appropriate.
	［説明］	セクション属性の記述が間違っています。
	［対処方法］	セクション属性を記述し直してください。
E0552132	［メッセージ］	Section has already determined as attribute.
	［説明］	セクションは既に相対属性に確定しています。制御命令 ".ORG" は記述できません。
	［対処方法］	セクションの属性を確認してください。
E0552133	［メッセージ］	Section attribute is not defined.
	［説明］	セクションの属性が未定義です。このセクション内では制御命令 ".ALIGN" は記述できません。
	［対処方法］	制御命令 ".ALIGN" は、絶対アドレス属性セクション又は ALIGN 指定のある相対アドレス属性セクション内に記述してください。
E0552134	［メッセージ］	Section name is missing.
	［説明］	セクション名がありません。
	［対処方法］	オペランドにセクション名を記述してください。

E0552135	［メッセージ］	'ALIGN' is multiple specified in '.SECTION'.
	［説明］	.SECTION 定義行に複数の 'ALIGN ' 指定があります。
	［対処方法］	余分な 'ALIGN' 指定を削除してください。
E0552136	［メッセージ］	Section type is multiple specified.
	［説明］	セクション定義行には、"CODE"、"DATA"、"ROMDATA" の指定は 1 つだけ記述してください。
E0552137	［メッセージ］	Too many operand.
	［説明］	オペランドが余分にあります。
	［対処方法］	オペランドの記述内容を確認してください。
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* " : " *アライメント数* "
	［説明］	" *オプション* " で指定した " *アライメント数* " は無効な値です。
	［対処方法］	1、2、4、8、16、または 32 を指定してください。
E0562017	［メッセージ］	Cannot output " *セクション* " specified in option " *オプション* "
	［説明］	" *オプション* " で指定した " *セクション* " のコードの一部を出力できません。命令コードのエンディアンを変換したことにより、" *セクション* " 内命令コードの一部が非連続となりました。
	［対処方法］	非連続部分の命令コードが属しているセクションは、リンク・マップからセクションアドレスを 4 バイト境界で確認のうえ、出力するセクションがどのセクションとエンディアン変換を行っているか確認してください。
E0562020	［メッセージ］	Duplicate file specified in option " *オプション* " : " *ファイル* "
	［説明］	" *オプション* " 指定で同じファイルを二度指定しました。
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］	" *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 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。
E0562140	［メッセージ］	Symbol/section " *名前* " redefined in option " *オプション* "
	［説明］	" *オプション* " で指定したシンボル、セクションはすでに定義されています。
E0562141	［メッセージ］	Module " *モジュール* " redefined in option " *オプション* "
	［説明］	" *オプション* " で指定したモジュールはすでに登録されています。
E0562142	［メッセージ］	Interrupt number " *ベクタ番号* " of " *セクション* " has multiple definition
	［説明］	ベクタ・テーブル " *セクション* " の、ベクタ番号定義が複数入力されました。ベクタ番号には、ひとつのアドレスしか設定できません。
	［対処方法］	ソース・ファイルの記述を見直してください。
E0562200	［メッセージ］	Illegal object file : " *ファイル* "
	［説明］	ELF フォーマット以外を入力しました。
E0562201	［メッセージ］	Illegal 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。
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。
E0562300	［メッセージ］	Duplicate symbol " *シンボル* " in " *ファイル* "
	［説明］	" *シンボル* " は重複しています。
E0562301	［メッセージ］	Duplicate module " *モジュール* " in " *ファイル* "
	［説明］	" *モジュール* " は重複しています。
E0562310	［メッセージ］	Undefined external symbol " *シンボル* " referenced in " *ファイル* "
	［説明］	" *ファイル* " 内で未定義の " *シンボル* " を参照しています。
E0562311	［メッセージ］	Section " *セクション1* " cannot refer to overlaid section : " *セクション2- シンボル* "
	［説明］	同一アドレスを指定したオーバレイセクション間でシンボル参照がありました。
	［対処方法］	" *セクション1* " と " *セクション2* " を同じアドレスに割り付けないでください。

E0562320	［メッセージ］	Section address overflowed out of range : " *セクション*"
	［説明］	" *セクション*" のアドレスが使用可能なアドレス範囲を越えました。
E0562321	［メッセージ］	Section " *セクション1*" overlaps section " *セクション2*"
	［説明］	" *セクション1*" と " *セクション2*" のアドレスが重複しました。
	［対処方法］	start オプションのアドレス指定を変更してください。 リンカで配置しようとした際に " *セクション2*" が " *セクション1*" の領域に重なってしまっています。 " *セクション1*" もしくは " *セクション2*" に対して配置指定されているアドレスを変更してください。
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。 CC-RX(ビルドツール) プロパティの共通オプションタブ内で CPU- 分岐幅を指定しない (-branch=32)(デフォルトは 24bit 以内であるとしてコンパイル (-branch=24)) に変更してみてください。 または「11.4 最適化リンケージエディタにおいて最適化有効時の E0562330 エラー発生の回避」に該当していないか確認してください。
E0562331	［メッセージ］	Division by zero in relocation value calculation : " *ファイル*"-" *セクション*"-" *オフセット*"
	［説明］	リロケーション演算に 0 除算が発生しました。
	［対処方法］	コンパイル、アセンブル・リストで、" *セクション*" の " *オフセット*" 位置の演算に問題がないか確認してください。
E0562332	［メッセージ］	Relocation value is odd number : " *ファイル*"-" *セクション*"-" *オフセット*"
	［説明］	リロケーション演算結果が奇数になりました。
	［対処方法］	コンパイル、アセンブル・リストで、" *セクション*" の " *オフセット*" 位置の演算に問題がないか確認してください。
E0562340	［メッセージ］	Symbol name " *ファイル*"-" *セクション*"-" *シンボル*" is too long
	［説明］	" *セクション*" 内の " *シンボル*" の文字数がアセンブラの翻訳限界を越えました。
	［対処方法］	シンボル・アドレス・ファイルを出力する場合は、アセンブラの翻訳限界文字数以下になるようなシンボル名としてください。
E0562366	［メッセージ］	" *セクション*" specified in option " *オプション*" was moved other area.
	［説明］	" *オプション*" で指定した " *セクション*" は -cpu=stride 指定により分割されました。
	［対処方法］	分割したセクションに対して " *オプション*" は指定できません。 -cpu=stride を指定しないか、分割を回避できるように -start オプションを見直してください。
E0562403	［メッセージ］	Fast interrupt register in " *ファイル*" conflicts with that in another file
	［説明］	" *ファイル*" 内で指定した高速割り込み用汎用レジスタ番号が、他ファイルと統一されていません。
	［対処方法］	高速割り込み用汎用レジスタ番号を他ファイルに合わせて、再度コンパイルして下さい。

E0562404	［メッセージ］	Base register " *ベースレジスタ種別* " in " *ファイル* " conflicts with that in another file
	［説明］	" *ファイル* " 内で指定した " *ベースレジスタ種別* " 用のレジスタ番号が、他ファイルと統一されていません。
	［対処方法］	ベース・レジスタ番号を他ファイルに合わせて、再度コンパイルして下さい。
E0562405	［メッセージ］	Option " *コンパイルオプション* " in " *ファイル* " conflicts with that in another files
	［説明］	" *ファイル* " について指定した " *コンパイルオプション* " の指定が入力ファイル間で統一されていません。
	［対処方法］	コンパイル・オプションを見直してください。
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) ～ (2) 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。
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、他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。
E0563602	［メッセージ］	"character string" option requires *Edition*.
	［説明］	" 文字列 " オプションの使用には *Edition* が必要です。
E0572000	［メッセージ］	Invalid option : " *オプション* "
	［説明］	" *オプション* " はサポートしていません。
E0572200	［メッセージ］	Illegal object file : " *ファイル* "
	［説明］	ELF フォーマット以外を入力しました。
E0572500	［メッセージ］	Cannot find file : " *ファイル* "
	［説明］	ライブラリとして指定した " *ファイル* " がありません。
E0572501	［メッセージ］	" *インスタンス* " has been referenced as both an explicit specialization and a generated instantiation
	［説明］	すでに定義が存在しているインスタンスに対して、インスタンス生成を要求しています。
	［対処方法］	" *インスタンス* " を使用しているファイルに対して、form=relocate でリロケータブル・ファイルを作成していないか確認してください。
E0572502	［メッセージ］	" *インスタンス* " assigned to " *ファイル 1* " and " *ファイル 2* "
	［説明］	" *ファイル 1* " と " *ファイル 2* " に " *インスタンス* " 定義が重複しています。
	［対処方法］	" *インスタンス* " を使用しているファイルに対して、form=relocate でリロケータブル・ファイルを作成していないか確認してください。
E0573005	［メッセージ］	Instantiation loop
	［説明］	インスタンス生成処理がループしています。入力ファイル名が別ファイルのファイル名と一致している可能性があります。
	［対処方法］	拡張子を除いたファイル名が一致しないようにファイル名を変更してください。
E0573007	［メッセージ］	Cannot create instantiation request file " *ファイル* "
	［説明］	インスタンス生成処理用の中間ファイルを作成できません。
	［対処方法］	オブジェクト作成フォルダ以下のアクセス権が正しいか確認してください。
E0573008	［メッセージ］	Cannot change to directory " *フォルダ* "
	［説明］	" *フォルダ* " に移動できません。
	［対処方法］	" *フォルダ* " が存在するか確認してください。
E0573009	［メッセージ］	File " *ファイル* " is read-only
	［説明］	" *ファイル* " は読み取り専用です。
	［対処方法］	アクセス権を変更してください。
E0573300	［メッセージ］	Cannot open file : " *ファイル* "
	［説明］	" *ファイル* " をオープンできません。
	［対処方法］	ファイル名およびアクセス権が正しいか、確認してください。
E0573303	［メッセージ］	Cannot read file : " *ファイル* "
	［説明］	" *ファイル* " を読めません。空ファイルを入力したか、ディスク容量に空きがない可能性があります。

E0573310	［メッセージ］	Cannot open temporary file
	［説明］	中間ファイルをオープンできません。HLNK_TMP 指定が正しいか確認してください。またはディスク容量に空きがない可能性があります。
E0573320	［メッセージ］	Memory overflow
	［説明］	最適化リンケージエディタが内部で使用するメモリが不足しています。
	［対処方法］	メモリを増やしてください。
E0573500	［メッセージ］	Bad instantiation request file -- instantiation assigned to more than one file
	［説明］	インスタンス生成処理用の中間ファイルに誤りがあります。
	［対処方法］	リンク対象ファイルを再コンパイルしてください。
E0573505	［メッセージ］	corrupted template information file or instantiation request file
	［説明］	テンプレート処理用中間ファイル、またはインスタンス生成処理用の中間ファイルのデータが誤っています。これらのファイルの編集はしないでください。
E0592001	［メッセージ］	Multiple input files are not allowed.
	［対処方法］	複数のファイルをコンバートする際は、リスト・ファイルを使用してください。
E0592002	［メッセージ］	Multiple output files are not allowed.
	［対処方法］	複数のファイルをコンバートする際は、リスト・ファイルを使用してください。
E0592003	［メッセージ］	List file is specified more than once.
	［対処方法］	1 つのリスト・ファイルにまとめてください。
E0592004	［メッセージ］	Invalid argument for the "*option*" option.
	［対処方法］	引数を確認してください。
E0592005	［メッセージ］	The "*option*" option can not have an argument.
	［説明］	引数の指定できない "*option*" オプションに引数が指定されています。
E0592006	［メッセージ］	The "*option*" option requires an argument.
	［説明］	引数の必要な "*option*" オプションに引数が指定されてません。
E0592007	［メッセージ］	The "*option*" option is specified more than once.
	［説明］	"*option*" オプションは、同時に 1 つしか指定できません。
E0592008	［メッセージ］	Requires an output file.
	［説明］	指定された入力ファイルに対応する出力ファイルが指定されていません。
E0592010	［メッセージ］	Failed to open an output file "*file*".
E0592013	［メッセージ］	Failed to delete a temporary file "*file*".
E0592015	［メッセージ］	Failed to close an input file "*file*".
E0592016	［メッセージ］	Failed to write an output file "*file*".
E0592018	［メッセージ］	Failed to open an list file "*file*".
	［対処方法］	存在するリスト・ファイルが正しく指定されているか確認してください。
E0592019	［メッセージ］	Syntax error in list file "*file*".
	［説明］	リスト・ファイル "*file*" の記述が正しくありません。
E0592020	［メッセージ］	Failed to read a list file "*file*".

E0592101	［メッセージ］	Unknown character '*C*'.
	［説明］	変換前の C 言語ソース・ファイルに C 言語で許可されていない文字があるため変換できません。
	［対処方法］	C 言語ソース・ファイルを編集して、構文エラーを修正してください。
E0592102	［メッセージ］	Illegal syntax in *string*.
	［説明］	変換前の C 言語ソース・ファイルで許可されていない文法があるため変換できません。
	［対処方法］	C 言語ソース・ファイルを編集して、構文エラーを修正してください。
E0592201	［メッセージ］	Illegal syntax.
	［説明］	変換前のアセンブリ言語ソース・ファイルに構文の誤りがあるため変換できません。
	［対処方法］	アセンブリ言語ソース・ファイルを編集して、構文エラーを修正してください。
E0593002	［メッセージ］	"-Xsfg_size_tidata_byte" size larger than "-Xsfg_size_tidata" size.
	［対処方法］	“-Xsfg_size_tidata_byte” サイズを、“-Xsfg_size_tidata” サイズ以下に設定するか、“-Xsfg_size_tidata” サイズを “-Xsfg_size_tidata_byte” サイズ以上に設定してください。
E0593003	［メッセージ］	Can not Read Symbol Information.
	［説明］	メモリ上のシンボル解析情報が存在しないか、壊れているため読み込むことができません。
	［対処方法］	CX オプションを確認後、リビルドを行ってください。
E0593004	［メッセージ］	Can not Write the SFG file.
	［説明］	容量またはユーザ権限の問題が考えられます。
	［対処方法］	書き込み先の容量とユーザ権限の確認をお願いします。
E0594000	［メッセージ］	Cannot find project file(*file name*).
	［説明］	プロジェクト・ファイルが存在しません。
	［対処方法］	ファイルが存在するか確認してください。
E0594001	［メッセージ］	Project file read error(*file name*).
	［説明］	プロジェクト・ファイルの読み込み時にエラーが発生しました。 プロジェクト・ファイルが読み込み禁止である可能性があります。
E0594002	［メッセージ］	Illegal format in project file(*file name*).
	［説明］	プロジェクト・ファイルが、不正なフォーマットです。
	［対処方法］	プロジェクト・ファイルで不正なフォーマットを発見した場合に表示されます。該当部を修正するか、プロジェクトを作り直してください。

10.5.3 致命的エラー

表 10.4 致命的エラー

F0511128	[メッセージ]	file "*file-name*" is not found.
	[説明]	ライブラリ " *ライブラリ名* " が見つかりません。
F0512003	[メッセージ]	Too many errors.
F0520003	[メッセージ]	#include file "*file*" includes itself.
	[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]	Out of memory.
	[対処方法]	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、再度コンパイルし直してください。
F0520005	[メッセージ]	Could not open source file "*file*".
	[説明]	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]	Expected a file name.
	[説明]	ファイル名がありません。
F0520016	[メッセージ]	*ファイル名* is not a valid source file name.
	[説明]	" *ファイル名* " は正しいソース・ファイル名ではありません。
F0520035	[メッセージ]	#error directive: *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]	Program too large or complicated to compile.
	[説明]	プログラムはコンパイルするのに大きすぎるか複雑すぎます。
F0520163	[メッセージ]	Could not open temporary file xxx.
	[説明]	テンポラリ・ファイル xxx がオープンできません。
F0520164	[メッセージ]	Name of directory for temporary files is too long (*xxx*).
	[説明]	テンポラリ・ファイル用のフォルダ名が長すぎます (xxx)。
F0520182	[メッセージ]	Could not open source file *xxx* (no directories in search list).
	[説明]	ソース・ファイル *xxx* を開くことができません。サーチ・リストにフォルダがありません。
F0520189	[メッセージ]	Error while writing *ファイル名* file.
	[説明]	ファイル " *ファイル名* " の書き込み中にエラーが生じました。
F0520190	[メッセージ]	Invalid intermediate language file.
	[説明]	不正な中間言語ファイルです。
F0520219	[メッセージ]	Error while deleting file *ファイル名* .
	[説明]	ファイル " *ファイル名* " の削除中にエラーが生じました。
F0520542	[メッセージ]	Could not create instantiation request file *名前* .
	[説明]	テンプレートを実体化するのに使用するファイル " *名前* " を作成することができませんでした。

F0520563	［メッセージ］	Invalid preprocessor output file .
	［説明］	不正なプリプロセッサ出力ファイルです。
F0520564	［メッセージ］	Cannot open preprocessor output file.
	［説明］	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］	Invalid option: *option*
F0520641	［メッセージ］	*xxx* is not a valid directory.
	［説明］	*xxx* は正しいフォルダではありません。
F0520642	［メッセージ］	Cannot build temporary file name.
	［説明］	テンポラリ・ファイル名を生成できません。
F0520869	［メッセージ］	Could not set locale *xxx* to allow processing of multibyte characters.
	［説明］	マルチバイト・キャラクタを処理するロケール *xxx* がセットできません。
F0520919	［メッセージ］	Invalid output file: *xxx*
	［説明］	不正な出力ファイルです : *xxx*。
F0520920	［メッセージ］	Cannot open output file: *xxx*
	［説明］	出力ファイルがオープンできません : *xxx*。
F0520926	［メッセージ］	Cannot open definition list file: *xxx*
	［説明］	定義リスト・ファイルがオープンできません : *xxx*。
F0521083	［メッセージ］	Exported template file *xxx* is corrupted.
	［説明］	エクスポートされたテンプレート・ファイルは破損しています。
F0521151	［メッセージ］	Mangled name is too long.
	［説明］	マングル名が長すぎます。
F0521335	［メッセージ］	Cannot open predefined macro file: *xxx*
	［説明］	既定義マクロ・ファイルがオープンできません : *xxx*。
F0521336	［メッセージ］	Invalid predefined macro entry at line *行数* : *行数 2*
	［説明］	不正な定義済みマクロの entry 宣言が " *行数* " にあります。
F0521337	［メッセージ］	Invalid macro mode name *xxx.*
	［説明］	不正なマクロ・モード名です *xxx*。
F0521338	［メッセージ］	Incompatible redefinition of predefined macro *xxx*.
	［説明］	互換性のない定義済みマクロの再定義です。
F0523029	［メッセージ］	Cannot open rule file.
	［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］	Incorrect description *ファイル名* in rule file
	［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］	Rule *ルール番号* is unsupported.
	［説明］	サポートしていないルール番号を指定しました。

F0523073	［メッセージ］	" *命令セットアーキテクチャ名* " does not support this intrinsic function.
	［説明］	サポートされていない組み込み関数を使用しています。-isa オプションの指定を確認してください。
F0523088	［メッセージ］	Bit position is out of range.
	［説明］	ビット位置が範囲外の値です。
F0523129	［メッセージ］	" *オプション名* " option is necessary for use of " *機能* ".
	［説明］	この機能を使用するにはオプションを指定する必要があります。
F0523300	［メッセージ］	Cannot open internal file.
	［説明］	コンパイラが内部で生成する中間ファイルをオープンすることができません。
F0523301	［メッセージ］	Cannot close internal file.
	［説明］	コンパイラが内部で生成する中間ファイルをクローズすることができません。
F0523302	［メッセージ］	Cannot write internal file.
	［説明］	中間ファイルの書き込み中にエラーが生じました。
F0530320	［メッセージ］	Duplicate symbol " *シンボル名* "
	［説明］	シンボル " *シンボル名* " が重複しています。
F0530800	［メッセージ］	Type of symbol " *シンボル名* " differs between files.
	［説明］	" *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］	Alignment of variable " *変数名* " differs between files.
	［説明］	" *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］	#pragma directive for symbol " *シンボル名* " differs between files.
	［説明］	" *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。
F0533021	［メッセージ］	Out of memory.
	［対処方法］	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］	Cannot close 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クローズすることができません。
F0533302	［メッセージ］	Cannot read an intermediate file.
	［説明］	中間ファイルの読み込み中にエラーが生じました。
F0533303	［メッセージ］	Cannot write to an intermediate file.
	［説明］	中間ファイルの書き込み中にエラーが生じました。
F0533306	［メッセージ］	Compilation was interrupted.
	［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］	Cannot read file " *ファイル名* ".
	［説明］	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］	Illegal stack access.
	［説明］	関数内で使用するスタックのサイズが 2G バイトを越えています。
F0540300	［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。
F0540301	［メッセージ］	Cannot close 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クローズすることができません。
F0540302	［メッセージ］	Cannot read an intermediate file.
	［説明］	中間ファイルの読み込み中にエラーが生じました。
F0540303	［メッセージ］	Cannot write to an intermediate file.
	［説明］	中間ファイルの書き込み中にエラーが生じました。
F0540400	［メッセージ］	Different parameters are set for the same #pramga " *識別子名* ".
	［説明］	#pragma " *識別子名* " に異なるパラメータが設定されています。
F0544302	［メッセージ］	Cannot read an intermediate file.
	［説明］	中間ファイルの読み込み中にエラーが生じました。
F0544802	［メッセージ］	The value of the parameter for the in-line function is outside the defined range.
	［説明］	組み込み関数の引数に指定した値が、関数で定義された範囲を越えています。
F0553000	［メッセージ］	Can't create file '*filename*'.
	［説明］	'*filename*' ファイルが生成できません。
	［対処方法］	ディレクトリ容量を確認してください。
F0553001	［メッセージ］	Can't open file '*filename*'.
	［説明］	'*filename*' ファイルがオープンできません。
	［対処方法］	ファイル名を確認してください。
F0553002	［メッセージ］	Can't write file '*filename*'.
	［説明］	'*filename*' ファイルに書き込むことができません。
	［対処方法］	ファイルのパーミッションを確認してください。
F0553003	［メッセージ］	Can't read file '*filename*'.
	［説明］	ファイルを読み込むことができません。
	［対処方法］	ファイルのパーミッションを確認してください。
F0553004	［メッセージ］	Can't create Temporary file.
	［説明］	テンポラリ・ファイルが生成できません。
	［対処方法］	カレントディレクトリ以外にテンポラリ・ファイルを作成するように、環境変数 'TMP_RX' にディレクトリを指定してください。
F0553005	［メッセージ］	Can't open Temporary file.
	［説明］	テンポラリ・ファイルがオープンできません。
	［対処方法］	'TMP_RX' で指定したディレクトリを確認してください。

F0553006	［メッセージ］	Can't read Temporary file.
	［説明］	テンポラリ・ファイルを読み込むことができません。
	［対処方法］	‘TMP_RX’ で指定したディレクトリを確認してください。
F0553007	［メッセージ］	Can't write Temporary file.
	［説明］	テンポラリ・ファイルに書き込むことができません。
	［対処方法］	‘TMP_RX’ で指定したディレクトリを確認してください。
F0553008	［メッセージ］	Illegal file name '*filename*'.
	［説明］	ファイル名が不正です。
	［対処方法］	ファイル名の記述規則に従ったファイル名を指定してください。
F0553016	［メッセージ］	Lacking cpu specification.
	［説明］	CPU の指定がされていません。
	［対処方法］	cpu オプション、または環境変数 CPU_RX で CPU を指定してください。
F0553100	［メッセージ］	Command line is too long.
	［説明］	コマンド行の文字数が多すぎます。
	［対処方法］	コマンドを入力し直してください。
F0553101	［メッセージ］	Invalid option '*xx*' is used.
	［説明］	無効なコマンドオプション *xx* を使用しています。
	［対処方法］	指定したオプションは存在しません。コマンドを入力し直してください。
F0553102	［メッセージ］	Ignore option '*xx*'.
	［説明］	無効なオプションが指定されています。
F0553103	［メッセージ］	Option '*xx*' is not appropriate.
	［説明］	コマンドオプション *xx* の記述が正しくありません。
	［対処方法］	コマンドオプションを指定し直してください。
F0553104	［メッセージ］	No input files specified.
	［説明］	入力ファイルの指定がありません。
	［対処方法］	入力ファイルを指定してください。
F0553105	［メッセージ］	Source files number exceed 80.
	［説明］	ファイルの数が 80 を越えています。
	［対処方法］	複数回にわけてアセンブルを実行してください。
F0553106	［メッセージ］	Lacking cpu specification.
	［説明］	CPU の指定がされていません。
	［対処方法］	cpu オプション、または環境変数 CPU_RX で CPU を指定してください。
F0553110	［メッセージ］	Multiple register base/fint_register.
	［説明］	base と fint_register オプションで指定レジスタが重複しています。
F0553111	［メッセージ］	Multiple register base/pid.
	［説明］	base と pid オプションで指定レジスタが重複しています。

F0553112	［メッセージ］	Multiple register base/nouse_pid_register.
	［説明］	base と nouse_pid_register オプションで指定レジスタが重複しています。
F0553113	［メッセージ］	Neither isa nor cpu is specified
	［説明］	-isa オプション、-cpu オプションのいずれの指定もありません。
F0553114	［メッセージ］	Both '-isa' option and '-cpu' option are specified
	［説明］	-isa オプションと -cpu オプションを同時に指定しています。
F0553115	［メッセージ］	The '-cpu' option and the '-fpu' option are inconsistent
	［説明］	-cpu（CPU_RX）の指定と -fpu の指定が矛盾しています。
F0553200	［メッセージ］	Error occurred in executing '*xxx*'.
	［説明］	*xxx* の実行でエラーが発生しました。
	［対処方法］	環境変数の設定を見直してください。
F0553201	［メッセージ］	Not enough memory.
	［説明］	メモリが足りません。
	［対処方法］	ファイルを分割して実行し直してください。または、メモリを増設してください。
F0553202	［メッセージ］	Can't find work dir.
	［説明］	ワークディレクトリが見つかりません。
	［対処方法］	環境変数 TMP_RX が正しく設定されているかを確認してください。
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
	［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。
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。 エラーで指摘している " *セクション*" が、RX で使用可能なアドレスの上限領域 (0xFFFFFFFF) を越えています。 なお、リンク時に最適化オプションを指定している場合は、リンク時の最適化が実施される前のリンク結果が RX で使用可能なアドレスの上限領域を越えてしまう場合もエラーを出力します。 ライブラリファイルを使用して参照されないシンボルをリンクしているときには、以下の方法で回避できる可能性があります。 ライブラリファイルに登録するオブジェクトファイルは、できるだけ関数単位 (1 関数、1 ファイル) になるようにファイルを分割してください。
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。
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。
F0563120	［メッセージ］	Illegal size of the absolute code section : " *セクション*" in " *ファイル*"
	［説明］	" *ファイル*" に存在する絶対アドレス・プログラム・セクション " *セクション*" のサイズが不正です。
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、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］	リンカが内部で使用するメモリが不足しています。
	［対処方法］	メモリを増やしてください。
F0563400	［メッセージ］	Cannot execute " *ロードモジュール* "
	［説明］	" *ロードモジュール* " を起動できません。
	［対処方法］	" *ロードモジュール* " のパスが設定されているか確認してください。
F0563410	［メッセージ］	Interrupt by user
	［説明］	標準入力端末から「(Ctrl)+C」キーによる割り込みを検出しました。
F0563420	［メッセージ］	Error occurred in " *ロードモジュール* "
	［説明］	" *ロードモジュール* " 実行中にエラーが発生しました。

F0563430	［メッセージ］	The total section size exceeded the limit 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 128K 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。
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。
F0578200	［メッセージ］	memory allocation fault
	［説明］	メモリが足りません。
F0578201	［メッセージ］	bad key *character* - use [dm(a|b)qr(a|b|u)txV]
	［説明］	*character* をキーとして指定することはできません。
F0578202	［メッセージ］	bad option *character* - use [cv]
	［説明］	*character* をオプションとして指定することはできません。
F0578203	［メッセージ］	bad option *string*
	［説明］	*string* をオプションとして指定することはできません。
F0578204	［メッセージ］	can not create file *file*
	［説明］	ファイル *file* を作成できません。
F0578205	［メッセージ］	file name *name*... is too long - limit is *number*
	［説明］	ファイル名 *name* の長さが限界を越えています。限界値は *number1* です。
F0578206	［メッセージ］	can not open file *file*
	［説明］	ファイル *file* をオープンできません。
F0578207	［メッセージ］	can not close file *file*
	［説明］	ファイル *file* をクローズできません。
F0578208	［メッセージ］	can not read file *file*
	［説明］	ファイル *file* からの読み込みができません。
F0578209	［メッセージ］	can not write file *file*
	［説明］	ファイル *file* への書き込みができません。
F0578210	［メッセージ］	can not seek file *file*
	［説明］	ファイル *file* をシークできません。
F0578212	［メッセージ］	can not nest command file *file*
	［説明］	コマンド・ファイル *file* がネストしています。ネストはできません。

F0578213	［メッセージ］	file is not *file*
	［説明］	*file* はライブラリ・ファイルではありません。
F0578214	［メッセージ］	malformed file *file*
	［説明］	ライブラリ・ファイル *file* の内容が破壊されているおそれがあります。
F0578215	［メッセージ］	can not find member *member*
	［説明］	ライブラリ・ファイル内にメンバ *member* が存在しません。
F0578216	［メッセージ］	symbol table limit error *file* (*number1*) - limit is *number2*
	［説明］	ライブラリ・ファイル *file* において、シンボルの個数 *number1* が限界を越えました。限界値は *number2* です。
F0578217	［メッセージ］	symbol table error *file*
	［説明］	ライブラリ・ファイル *file* において、ライブラリ・シンボル・テーブルの作成に失敗しました。
F0578218	［メッセージ］	string table error *file*
	［説明］	ライブラリ・ファイル *file* において、ライブラリ・ストリング・テーブルの内容が破壊されているおそれがあります。
F0578219	［メッセージ］	*file* has no member
	［説明］	ライブラリ・ファイル *file* 内にメンバが存在しません。
F0578220	［メッセージ］	version error *file*
	［説明］	指定されたファイル *file* の形式の版が本ライブラリアンの扱うことのできる版ではありません。
F0578221	［メッセージ］	can not read header *file*
	［説明］	ライブラリ・ファイル *file* のヘッダの読み込みができません。
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。
F0593113	［メッセージ］	Neither isa nor cpu is specified
	［説明］	-isa オプション、-cpu オプションのいずれの指定もありません。
F0593114	［メッセージ］	Both '-isa' option and '-cpu' option are specified
	［説明］	-isa オプションと -cpu オプションを同時に指定しています。
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。

10.5.4 インフォメーション

表 10.5 インフォメーション

M0520009	[メッセージ]	Nested comment is not allowed.
	[説明]	コメントのネスティングは許されていません。
M0520018	[メッセージ]	Expected a ")".
	[説明]	")" がありません。
M0520111	[メッセージ]	Statement is unreachable.
	[説明]	文は実行されません。
M0520128	[メッセージ]	Loop is not reachable from preceding code.
	[説明]	ループはその前のコードから到達しません。
M0520174	[メッセージ]	Expression has no effect.
	[説明]	式は作用しません。
M0520177	[メッセージ]	*種別* " *シンボル名* " was declared but never referenced
	[説明]	*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。
M0520193	[メッセージ]	Zero used for undefined preprocessing identifier *xxx*.
	[説明]	定義されていないプリプロセッサ識別子 *xxx* に対して 0 が使用されます。
M0520237	[メッセージ]	Selector expression is constant.
	[説明]	選択式が定数です。
M0520261	[メッセージ]	Access control not specified (" *名前* " by default).
	[説明]	基底クラスのアクセス制御指定がありません。アクセス制御指定 " *名前* " が仮定されます。
M0520324	[メッセージ]	Duplicate friend declaration.
	[説明]	フレンド宣言が重複して指定されています。
M0520381	[メッセージ]	Extra ";" ignored.
	[説明]	余分な ";" を無視しました。
M0520399	[メッセージ]	*名前* has an operator new *xxx*() but no default operator delete *xxx*().
	[説明]	" *名前* " が operator new を持ちますが、デフォルトの operator delete を持ちません。
M0520400	[メッセージ]	*名前* has a default operator delete *xxx*() but no operator new *xxx*().
	[説明]	" *名前* " がデフォルトの operator delete を持ちますが、operator new を持ちません。
M0520479	[メッセージ]	*名前* redeclared "inline" after being called.
	[説明]	関数が呼ばれたあとに inline" *名前* " を宣言しています。以降 inline 指定を有効にします。
M0520487	[メッセージ]	Inline *名前* cannot be explicitly instantiated.
	[説明]	インライン関数 " *名前* " を実体化することはできません。
M0520534	[メッセージ]	Use of a local type to specify an exception.
	[説明]	ローカルな型を使用した例外処理が指定されています。

M0520535	［メッセージ］	Redundant type in exception specification.
	［説明］	例外処理中に冗長な型の指定があります。
M0520549	［メッセージ］	*symbol* is used before its value is set.
	［説明］	*symbol* は値が設定される前に使用されました。
M0520550	［メッセージ］	"*symbol*" was set but never used.
	［説明］	" *シンボル名* " は設定されていますが利用されていません。
M0520618	［メッセージ］	Struct or union declares no named members.
	［説明］	構造体か共用体に名前のないメンバがあります。
M0520652	［メッセージ］	Calling convention is ignored for this type.
	［説明］	この型に対する呼び出し規約は無視されます。
M0520678	［メッセージ］	Call of "*symbol*" cannot be inlined.
	［説明］	*symbol* の呼び出しはインライン展開できません。
M0520679	［メッセージ］	*symbol* cannot be inlined.
	［説明］	*symbol* はインライン展開できません。
M0520815	［メッセージ］	Type qualifier on return type is meaningless.
	［説明］	返却型に対する型指定は意味がありません。
M0520826	［メッセージ］	*名前* was never referenced.
	［説明］	関数の引数 " *名前* " は参照されません。
M0520831	［メッセージ］	Support for placement delete is disabled.
	［説明］	operator delete 関数の型が正しくありません。処理を継続します。
M0520863	［メッセージ］	Effect of this "#pragma pack" directive is local to *xxx*.
	［説明］	#pragma pack ディレクティブの影響はシンボル内にとどまります。
M0520866	［メッセージ］	Exception specification ignored.
	［説明］	例外指定は無視されます。
M0520949	［メッセージ］	Specifying a default argument on this declaration is nonstandard.
	［説明］	この宣言にデフォルト引数を指定するのは標準形式ではありません。
M0521348	［メッセージ］	Declaration hides "*symbol*".
	［説明］	宣言は "*symbol*" を隠します。
M0521353	［メッセージ］	*シンボル* has no corresponding member operator delete *xxx* (to be called if an exception is thrown during initialization of an allocated object).
	［説明］	" *シンボル* " は new オペレータの対となる delete オペレータを持ちません（取得したオブジェクトの初期化時に例外が発生した場合に呼ばれます）。
M0521380	［メッセージ］	Virtual *xxx* was not defined (and cannot be defined elsewhere because it is a member of an unnamed namespace).
	［説明］	仮想関数の定義がありません。また、無名空間のメンバであるため、それ以外の場所で定義することができません。

M0521381	［メッセージ］	Carriage return character in source line outside of comment or character/string literal.
	［説明］	改行文字がコメントまたは文字列リテラル以外のところにあります。
M0523009	［メッセージ］	This pragma has no effect.
	［説明］	この #pragma は無効です。
M0523028	［メッセージ］	Rule *ルール番号*: *内容*
	［説明］	MISRA-C:2004 の*ルール番号*と*内容*の該当箇所を検出しました。
M0523033	［メッセージ］	Precision lost.
	［説明］	代入式において、右辺の式の値を左辺の型へ変換する時に、精度が失われる可能性があります。
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。
M0560002	［メッセージ］	Symbol " *シンボル* " created by optimization " *最適化* "
	［説明］	" *最適化* " の最適化によって、" *シンボル* " を作成しました。
M0560004	［メッセージ］	" *ファイル* "-" *シンボル* " deleted by optimization
	［説明］	symbol_delete の最適化によって、" *ファイル* " 内の " *シンボル* " を削除しました。
M0560005	［メッセージ］	The offset value from the symbol location has been changed by optimization " *ファイル* "-" *セクション* "-" *シンボル* ± offset"
	［説明］	" *シンボル* ± offset" の範囲で最適化によるサイズ変更があったため offset 値を変更しました。問題ないか確認してください。offset 値の変更を抑止したい場合は、" *ファイル* " のアセンブル時に goptimize オプション指定を外してください。
M0560100	［メッセージ］	No inter-module optimization information in " *ファイル* "
	［説明］	" *ファイル* " 内にモジュール間最適化情報がありません。" *ファイル* " をモジュール間最適化の対象外にします。モジュール間最適化の対象にする場合は、コンパイル、アセンブル時に goptimize オプションを指定してください。
M0560101	［メッセージ］	No stack information in " *ファイル* "
	［説明］	" *ファイル* " 内にスタック情報がありません。" *ファイル* " はアセンブラ出力ファイルの可能性があります。リンカが出力するスタック情報ファイルに当該ファイルの内容は含まれません。
M0560102	［メッセージ］	Stack size " *サイズ* " specified to the undefined symbol " *シンボル* " in " *ファイル* "
	［説明］	" *ファイル* " 内の未定義シンボル " *シンボル* " に、スタックサイズ " *サイズ* " が指定されています。
M0560103	［メッセージ］	Multiple stack sizes specified to the symbol " *シンボル* "
	［説明］	" *シンボル* " は、複数のスタックサイズが指定されています。
M0560300	［メッセージ］	Mode type " *モード種別 1* " in " *ファイル* " differ from " *モード種別 2* "
	［説明］	異なるモード種別のファイルを入力しました。
M0560400	［メッセージ］	Unused symbol " *ファイル* "-" *シンボル* "
	［説明］	" *ファイル* " 内の " *シンボル* " は使用されていません。
M0560500	［メッセージ］	Generated CRC code at " *アドレス* "
	［説明］	" *アドレス* " に CRC コードを出力しました。

M0560510	［メッセージ］	Section " *セクション* " was moved other area specified in option "cpu=< *メモリ属性* >"
	［説明］	セクションを分割せずに cpu=< *メモリ属性* > にしたがって " *セクション* " を配置しました。
M0560511	［メッセージ］	Sections " *セクション名* "," *分割後のセクション名* " are Non-contiguous
	［説明］	" *セクション名* " のセクションを分割し、" *分割後のセクション名* " のセクションを生成しました。
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］	"*path*" specified by the "*character string*" option is a file. Specify a folder.
	［説明］	" *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］	The folder "*folder*" specified by the "*character string*" option is not found.
	［説明］	" *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］	The "*character string2*" option is ignored when the "*character string1*" option is specified at the same time.
	［説明］	" *文字列 1*" オプションが指定されたので " *文字列 2*" オプションは無視しました。
W0511143	［メッセージ］	The "-Xfloat" option is ignored because specified device does not have FPU.
	［説明］	FPU を持っていないデバイスが指定されたので、"-Xfloat" オプションを無視しました。
W0511144	［メッセージ］	"-C" option and "-Xcommon" option is mismatch. Instruction set by "*character string1*" option is ignored. Create common object for "*character string2*" instruction set.
	［説明］	"-C" オプションと "-Xcommon" オプションが一致しません。" *文字列 1*" オプションの命令セットを無視します。" *文字列 2*" 命令セットのコモンオブジェクトを生成します。
W0511146	［メッセージ］	"*symbol name*" specified in the "*character string*" option is not allowed for a preprocessor macro. Recognized only as an assembler symbol.
	［説明］	" *文字列* " オプションで指定された " *シンボル名* " は C 言語のマクロでは使用できません。アセンブラのみに指定されたとみなします。
W0511147	［メッセージ］	The "*character string*" option is specified more than once. The latter is valid.
	［説明］	" *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］	The "*character string2*" option is ignored when the "*character string1*" option and the "*character string2*" option are inconsistent.
	［説明］	" *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］	The "*character string2*" option is ignored when the "*character string1*" option is not specified.
	［説明］	" *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］	Optimization itemoptions were cleared when "-O*character string*" option is specified. Optimization itemoptions need to specify after "-O*character string*" option.
	［説明］	"-O *文字列* " が指定されたので最適化詳細オプションはクリアされました。最適化詳細オプションは "-O *文字列* " の後に指定してください。
W0511156	［メッセージ］	Device file is not found in the folder specified by the "-Xdev_path" option.
	［説明］	"-Xdev_path" オプションで指定されたフォルダ上に、デバイス・ファイルが見つかりません。
W0511164	［メッセージ］	Dupulicate file name. "*file-name*".
	［説明］	同じファイル名 " *ファイル名* " が複数指定されています。

W0511166	［メッセージ］	"*macro name*" is not a valid predefined macro name.
	［説明］	マクロ名 " *マクロ名* " はプリディファインドマクロではありません。undefine オプションの指定を無効とします。
W0511168	［メッセージ］	"*option-name*" option has no effect in this version.
	［説明］	本バージョンでは無効なオプションです。
W0511169	［メッセージ］	" *内容* " is not valid in "*language specifications*"
	［説明］	表示された " *内容* " は " *言語仕様*（C、または C++）" では無効です。
W0511170	［メッセージ］	"*option-name*" option is ignored due to the specification of another option.
	［説明］	このオプションは他のオプションの指定により無視されます。
W0511171	［メッセージ］	" *内容* " is ignored in "*language specifications*".
	［説明］	表示された " *内容* " は " *言語仕様*（C、または C++）" では無視されます。
W0511172	［メッセージ］	Nothing to compile, assemble or link.(input and output combination)
	［説明］	コンパイル、アセンブルまたはリンク処理のいずれも行う必要がありません。
	［対処方法］	入力ファイルの構成と output オプション指定の組み合わせを確認してください。
W0511179	［メッセージ］	The evaluation version is valid for the remaining *number* days.
	［説明］	この評価版は残り *数字* 日間有効です。
W0511180	［メッセージ］	The evaluation period of *バージョン* has expired.
	［説明］	*バージョン*の評価期間の有効期限が切れています。
W0511185	［メッセージ］	The trial period for the features of the Professional edition expires in *数字* days. Please consider purchasing the product of Professional edition.
	［説明］	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。
W0519999	［メッセージ］	The "*option-name*" option is not implemented.
	［説明］	" *オプション名* " オプションは実装されていません。
W0520001	［メッセージ］	Last line of file ends without a newline.
	［対処方法］	改行を追加してください。
W0520009	［メッセージ］	Nested comment is not allowed.
	［説明］	コメントのネスティングは許されていません。
	［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］	Unrecognized preprocessing directive.
	［説明］	不明な前処理指令があります。
W0520012	［メッセージ］	Parsing restarts here after previous syntax error.
	［説明］	前に構文エラーがあるため、ここより文法の解析を再開します。
W0520014	［メッセージ］	Extra text after expected end of preprocessing directive.
	［説明］	前処理指令の後に不正な文字があります。
W0520019	［メッセージ］	Extra text after expected end of number.
	［説明］	数値の後に不正な文字があります。

W0520021	[メッセージ]	Type qualifiers are meaningless in this declaration.
	[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]	Too many characters in character constant.
	[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]	Character value is out of range.
	[説明]	char 型の値が範囲を越えています。
W0520038	[メッセージ]	Directive is not allowed -- an #else has already appeared.
	[説明]	この前処理指令は許可されていません -- #else はすでにあります。
W0520039	[メッセージ]	Division by zero.
	[説明]	0 で除算を行いました。
W0520042	[メッセージ]	Operand types are incompatible (" *型 1*" and " *型 2*").
	[説明]	オペランドの型が適合しません (" *型 1*" と " *型 2*")。
W0520045	[メッセージ]	#undef may not be used on this predefined name.
	[説明]	既定義名に対して #undef を使用できません。
W0520046	[メッセージ]	*symbol* is predefined; attempted redefinition ignored.
	[説明]	*symbol* を再定義することはできません。
W0520047	[メッセージ]	Incompatible redefinition of macro "symbol".
	[説明]	マクロ "symbol" の適合しない再定義があります。
W0520054	[メッセージ]	Too few arguments in macro invocation.
	[説明]	マクロに対する引数が足りません。
W0520055	[メッセージ]	Too many arguments in macro invocation.
	[説明]	マクロに対する引数が多すぎます。
W0520061	[メッセージ]	Integer operation result is out of range.
	[説明]	整数演算の結果が範囲を越えました。
W0520062	[メッセージ]	Shift count is negative.
	[説明]	シフト数が負数です。ANSI-C では未定義の動作となります。
W0520063	[メッセージ]	Shift count is too large.
	[説明]	シフト数が多すぎます。
W0520064	[メッセージ]	Declaration does not declare anything.
	[説明]	この宣言は何も宣言できません。
W0520066	[メッセージ]	Enumeration value is out of "int" range.
	[説明]	enum の値が "int" の範囲を越えています。
W0520068	[メッセージ]	Integer conversion resulted in a change of sign.
	[説明]	整数変換で結果の符号が反転しました。
W0520069	[メッセージ]	Integer conversion resulted in truncation.
	[説明]	整数型の変換において、切り捨てが生じます。

W0520070	[メッセージ]	Incomplete type is not allowed.
	[説明]	不完全型は許されていません。
W0520076	[メッセージ]	Argument to macro is empty.
	[説明]	マクロに対する引数がありません。
W0520077	[メッセージ]	This declaration has no storage class or type specifier.
	[説明]	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]	Storage class is not first.
	[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]	Type qualifier specified more than once.
	[説明]	型修飾子が複数回指定されました。
W0520085	[メッセージ]	Invalid storage class for a parameter.
	[説明]	引数に対する記憶域クラスが不正です。
W0520086	[メッセージ]	Invalid storage class for a function.
	[説明]	関数に対する記憶域クラスが不正です。
W0520099	[メッセージ]	A declaration here must declare a parameter.
	[説明]	ここでの宣言は引数宣言でなければなりません。
W0520101	[メッセージ]	*xxx* has already been declared in the current scope.
	[説明]	*xxx* はすでにこのスコープで宣言されています。
W0520108	[メッセージ]	Signed bit field of length 1.
	[説明]	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]	Statement is unreachable.
	[説明]	文は実行されません。
W0520117	[メッセージ]	Non-void "symbol" should return a value.
	[説明]	void でない関数 " *シンボル* " は値を返す必要があります。
W0520118	[メッセージ]	A void function may not return a value.
	[説明]	void 関数は値を返せません。
W0520127	[メッセージ]	Expected a statement.
	[説明]	文がありません。
W0520128	[メッセージ]	Loop is not reachable from preceding code.
	[説明]	ループはその前のコードから到達しません。
W0520138	[メッセージ]	Taking the address of a register variable is not allowed.
	[説明]	レジスタ変数に対するアドレス演算子は許されていません。
W0520139	[メッセージ]	Taking the address of a bit field is not allowed.
	[説明]	ビット・フィールドに対するアドレス演算子は許されていません。
W0520140	[メッセージ]	Too many arguments in function call.
	[説明]	関数呼び出しに対する引数が多すぎます。

W0520144	[メッセージ]	A value of type " *型名 1*"" cannot be used to initialize an entity of type " *型名 2*".
	[説明]	型 " *型名 1*" の値は型 " *型名 2*" の実体の初期化には使用できません。
W0520147	[メッセージ]	Declaration is incompatible with " *宣言* ".
	[説明]	宣言は " *宣言* " と整合しません。
W0520152	[メッセージ]	Conversion of nonzero integer to pointer.
	[説明]	0 でない値がポインタに変換されました。
W0520157	[メッセージ]	Expression must be an integral constant expression.
	[説明]	式は整数定数式である必要があります。
W0520161	[メッセージ]	Unrecognized #pragma.
	[説明]	認識されない #pragma です。
W0520165	[メッセージ]	Too few arguments in function call.
	[説明]	関数呼び出しに引数が足りません。
W0520167	[メッセージ]	Argument of type " *型名 1*" is incompatible with parameter of type " *型名 2*".
	[説明]	" *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]	Pointer points outside of underlying object.
	[説明]	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]	Invalid type conversion.
	[説明]	不正な型変換です。
W0520172	[メッセージ]	External/internal linkage conflict with previous declaration.
	[説明]	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]	Floating-point value does not fit in required integral type.
	[説明]	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]	Expression has no effect.
	[説明]	式は作用しません。無効です。
W0520175	[メッセージ]	Subscript out of range.
	[説明]	添字が範囲を越えました。
W0520177	[メッセージ]	*種別* " *シンボル名* " was declared but never referenced
	[説明]	*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]	Right operand of "%" is zero.
	[説明]	"%" の右オペランドが 0 です。
W0520180	[メッセージ]	Argument is incompatible with formal parameter.
	[説明]	実引数が仮引数と整合しません。
W0520181	[メッセージ]	Argument is incompatible with corresponding format string conversion.
W0520185	[メッセージ]	Dynamic initialization in unreachable code.
	[説明]	実行されないコードの中に動的初期化があります。

W0520186	[メッセージ]	Pointless comparison of unsigned integer with zero.
	[説明]	符号なし整数と 0 の比較は無意味です。
W0520187	[メッセージ]	Use of "=" where "==" may have been intended.
	[説明]	"==" と思われる "=" の使用があります。
W0520188	[メッセージ]	Enumerated type mixed with another type.
	[説明]	列挙型に別の型が混在しています。
W0520191	[メッセージ]	Type qualifier is meaningless on cast type.
	[説明]	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]	Unrecognized character escape sequence.
	[説明]	認識されないエスケープ・シーケンスがあります。
W0520223	[メッセージ]	Function *xxx* declared implicitly.
	[説明]	関数 *xxx* は暗黙に宣言されました。
W0520224	[メッセージ]	The format string requires additional arguments.
	[説明]	フォーマット文字列にはさらなる引数が必要です。
W0520225	[メッセージ]	The format string ends before this argument.
	[説明]	フォーマット文字列に対して引数が多すぎます。
W0520226	[メッセージ]	Invalid format string conversion.
	[説明]	不正なフォーマット文字列です。
W0520229	[メッセージ]	Bt field cannot contain all values of the enumerated type.
	[説明]	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]	Declaration is not visible outside of function.
	[説明]	宣言は関数の外で見えません。
W0520232	[メッセージ]	Old-fashioned typedef of "void" ignored.
	[説明]	古い仕様の "void" typedef です。無視されました。
W0520236	[メッセージ]	Controlling expression is constant.
	[説明]	制御式が定数です。
W0520257	[メッセージ]	const "*symbol*" requires an initializer.
	[説明]	const 変数 "*symbol*" は初期化子が必要です。
W0520260	[メッセージ]	Explicit type is missing ("int" assumed).
	[説明]	明示的な型がありません。"int" として扱います。
W0520262	[メッセージ]	Not a class or struct name.
	[説明]	基底クラスで指定されたクラスまたは構造体がありません。
W0520280	[メッセージ]	Declaration of a member with the same name as its class.
	[説明]	クラス名と同じ名前のメンバ名を宣言しています。
W0520284	[メッセージ]	NULL reference is not allowed.
	[説明]	NULL へのリファレンスは許されません。指定された通りに式を評価します。

W0520296	[メッセージ]	Invalid use of non-lvalue array.
	[説明]	左辺値でない配列の不正な利用です。
W0520300	[メッセージ]	A pointer to a bound function may only be used to call the function.
	[説明]	メンバ関数へのポインタを関数呼び出し以外に使用しています。
W0520301	[メッセージ]	typedef name has already been declared (with same type).
	[説明]	typedef 名はすでに同じ型で宣言されています。
W0520326	[メッセージ]	Inline is not allowed.
	[説明]	inline は許されていません。
W0520335	[メッセージ]	Linkage specification is not allowed.
	[説明]	リンケージの指定は許されていません。
W0520368	[メッセージ]	*xxx* defines no constructor to initialize the following:
	[説明]	名前は初期化のためのコンストラクタを定義していません。
W0520370	[メッセージ]	*symbol* has an uninitialized const field.
	[説明]	*symbol* は const のフィールドが初期化されていません。
W0520375	[メッセージ]	Declaration requires a typedef name.
	[説明]	宣言は typedef 名を必要とします。
W0520377	[メッセージ]	"virtual" is not allowed.
	[説明]	virtual を指定することはできません。
W0520381	[メッセージ]	Extra ";" ignored.
	[説明]	余分な ";" を無視しました。
W0520382	[メッセージ]	In-class initializer for nonstatic member is nonstandard.
	[説明]	非スタティックなメンバを初期化するのは標準形式ではありません。
W0520414	[メッセージ]	Delete of pointer to incomplete class.
	[説明]	不完全型クラスへのポインタは削除されました。
W0520430	[メッセージ]	Returning reference to local temporary.
	[説明]	関数内にローカルな領域のリファレンスをリターン値にしています。
W0520494	[メッセージ]	Declaring a void parameter list with a typedef is nonstandard.
	[説明]	typedef を伴う void の引数リストの宣言は標準でありません。
W0520497	[メッセージ]	Declaration of %sq hides template parameter.
	[説明]	名前の宣言はテンプレート引数を隠蔽します。
W0520512	[メッセージ]	Type qualifier on a reference type is not allowed.
	[説明]	リファレンス型に const/volatile 修飾を指定することはできません。
W0520513	[メッセージ]	A value of type "*type1*" cannot be assigned to an entity of type "*type2*".
	[説明]	型 "*type1*" の値は型 "*type2*" の実体として代入できません。
W0520514	[メッセージ]	Pointless comparison of unsigned integer with a negative constant.
	[説明]	負の定数と unsigned 型の比較は意味がありません。

W0520520	[メッセージ]	Initialization with "{...}" expected for aggregate object.
	[説明]	集成体は "{...}" により初期化してください。
W0520522	[メッセージ]	Pointless friend declaration.
	[説明]	自分自身へのフレンド宣言をしています。
W0520523	[メッセージ]	"." used in place of "::" to form a qualified name.
	[説明]	“.” がスコープ解決子 “::” の代わりに使用されています。
W0520533	[メッセージ]	Handler is potentially masked by previous handler for type "type".
	[説明]	ハンドラは型 "type" の以前のハンドラに潜在的にマスクされます。
W0520541	[メッセージ]	Omission of exception specification is incompatible with previous *名前*.
	[説明]	例外処理の省略形が前の " *名前* " と合致しません。
W0520549	[メッセージ]	"*symbol*" is used before its value is set.
	[説明]	" *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]	"*symbol*" was set but never used.
	[説明]	" *シンボル名* " は設定されていますが利用されていません。
W0520552	[メッセージ]	Exception specification is not allowed.
	[説明]	例外処理指定は許されません。例外処理を無効にします。
W0520553	[メッセージ]	external/internal linkage conflict for "symbol".
	[説明]	external/internal のリンケージが不適合を起こしています "symbol"。
W0520554	[メッセージ]	*名前* will not be called for implicit or explicit conversions.
	[説明]	変換関数 " *名前* " は暗黙的にも明示的にも呼ばれることはありません。
W0520611	[メッセージ]	Overloaded virtual function *名前 1* is only partially overridden in *名前 2*.
	[説明]	*名前 1* のオーバーロード仮想関数は " *名前 2*" の中で一部の仮想関数だけが置き換えの対象になります。指定された通りに処理を継続します。
W0520617	[メッセージ]	Pointer-to-member-function cast to pointer to function.
	[説明]	メンバ関数ポインタを関数ポインタにキャストしています。
W0520618	[メッセージ]	Struct or union declares no named members.
	[説明]	構造体か共用体に名前のないメンバがあります。
W0520650	[メッセージ]	Calling convention specified here is ignored.
	[説明]	ここでの呼び出し規則指定は無視されました。
W0520657	[メッセージ]	Inline specification is incompatible with previous "*symbol*".
	[説明]	inline 指定子は前の "*symbol*" と整合しません。
W0520662	[メッセージ]	Call of pure virtual function.
	[説明]	純粋仮想関数が関数を呼び出しています。
W0520676	[メッセージ]	Using out-of-scope declaration of " *シンボル名* ".
	[説明]	" *シンボル名* " の宣言のスコープ外で使用されました。
W0520691	[メッセージ]	*xxx*, required for copy that was eliminated, is inaccessible.
	[説明]	コピーコンストラクタにアクセスできません。

W0520692	［メッセージ］	*xxx*, required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.
	［説明］	コピーコンストラクタを呼び出すことができません。
W0520708	［メッセージ］	Incrementing a bool value is deprecated.
	［説明］	bool 型の値をインクリメントしています。値をインクリメントして処理を継続します。
W0520720	［メッセージ］	Redeclaration of *xxx* is not allowed to alter its access.
	［説明］	名前の再宣言でアクセス指定を変更することはできません。前の宣言のアクセス指定を有効にします。
W0520722	［メッセージ］	Use of alternative token "<:" appears to be unintended.
	［説明］	2 文字表記 "<:" が使用されました。"[" と解釈します。
W0520723	［メッセージ］	Use of alternative token "%%:" appears to be unintended.
	［説明］	2 文字表記 "%:" が使用されました。"#" と解釈します。
W0520737	［メッセージ］	Using-declaration ignored -- it refers to the current namespace.
	［説明］	現在の namespace スコープの名前を using 宣言しています。using 宣言を無視します。
W0520748	［メッセージ］	Calling convention specified more than once.
	［説明］	呼び出し規則指定が複数回指定されています。
W0520760	［メッセージ］	*シンボル* explicitly instantiated more than once.
	［説明］	" *シンボル* " を具現化できませんでした。
W0520767	［メッセージ］	Conversion from pointer to smaller integer.
	［説明］	ポインタが幅の小さな整数に変換されました。
W0520780	［メッセージ］	Reference is to *シンボル 1* -- under old for-init scoping rules it would have been *シンボル 2*.
	［説明］	*" シンボル 1"* を参照しています。
W0520783	［メッセージ］	Empty comment interpreted as token-pasting operator "##".
	［説明］	空のコメントは字句連結オペレータ "##" と仮定します。
W0520794	［メッセージ］	Template parameter %sq may not be used in an elaborated type specifier.
	［説明］	class 指定にテンプレート引数を使用することはできません。class 指定を無効にしてテンプレートを有効にします。
W0520802	［メッセージ］	Specifying a default argument when redeclaring an unreferenced function template is nonstandard.
	［説明］	未参照の関数テンプレートを再宣言するときにデフォルト引数を指定しています。デフォルト引数を無視します。
W0520806	［メッセージ］	Omission of exception specification is incompatible with *名前* .
	［説明］	throw 例外指定の省略は " *名前* " の例外指定と合致しません。" *名前* " を有効にします。
W0520812	［メッセージ］	const object requires an initializer -- class *型* has no explicitly declared default constructor.
	［説明］	const 型オブジェクトには初期化指定が必要です。クラス " *型* " が明示的に宣言されたデフォルトコンストラクタを持ちません。

W0520815	［メッセージ］	Type qualifier on return type is meaningless.
	［説明］	返却型に対する型修飾子は意味がありません。
W0520825	［メッセージ］	Virtual inline *名前* was never defined.
	［説明］	仮想インラインメンバ関数 " *名前* " の定義がありません。
W0520826	［メッセージ］	*名前* was never referenced.
	［説明］	関数の引数 " *名前* " は参照されません。
W0520829	［メッセージ］	Double used for "long double" in generated C code.
	［説明］	double が "long double" として生成された C コードで使用されています。
W0520830	［メッセージ］	*xxx* has no corresponding operator delete*yyy* (to be called if an exception is thrown during initialization of an allocated object).
	［説明］	対応する operator delete がありません。
W0520831	［メッセージ］	Support for placement delete is disabled
	［説明］	operator delete 関数の型が正しくありません。
W0520836	［メッセージ］	Returning reference to local variable.
	［説明］	局所変数のリファレンスをリターン値に指定しています。指定された処理を継続します。
W0520837	［メッセージ］	Omission of explicit type is nonstandard ("int" assumed).
	［説明］	型指定がありません。int 型を仮定します。
W0520867	［メッセージ］	Declaration of "size_t" does not match the expected type "*type*".
	［説明］	size_t の宣言は期待された型 "*type*" と一致しません。
W0520870	［メッセージ］	Invalid multibyte character sequence.
	［説明］	不正な多バイト文字列です。
W0520902	［メッセージ］	Type qualifier ignored.
	［説明］	型修飾子を無視しました。
W0520912	［メッセージ］	Ambiguous class member reference -- *シンボル 1* used in preference to *シンボル 2*.
	［説明］	あいまいなクラスメンバの参照です。*シンボル 1* を *シンボル 2* に優先して参照します。
W0520925	［メッセージ］	Type qualifiers on function types are ignored.
	［説明］	関数型の型修飾子は無視されました。
W0520936	［メッセージ］	Redeclaration of *名前* alters its access.
	［説明］	" *名前* " の再宣言でアクセス指定を変更しています。再定義されたアクセス指定を有効にします。
W0520940	［メッセージ］	Missing return statement at end of non-void "*symbol*"..
	［説明］	void でない "*symbol*" に return 文がありません。
W0520941	［メッセージ］	Duplicate using-declaration of *名前* ignored.
	［説明］	using 宣言 " *名前* " を重複指定しています。重複した using 宣言を無効にします。

W0520942	［メッセージ］	enum bit-fields are always unsigned, but enum *xxx* includes negative enumerator.
	［説明］	列挙型のビットフィールドは常に unsigned です。しかし enum "type" は負の列挙子を持ちます。
W0520948	［メッセージ］	Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.
	［説明］	非標準形式のローカルクラスのフレンド宣言です。クラスの定義内に前方宣言がありません。
W0520951	［メッセージ］	Return type of function "main" must be "int".
	［説明］	main 関数の返却型は "int" である必要があります。
W0520959	［メッセージ］	Declared size for bit field is larger than the size of the bit field type; truncated to any-string bits.
	［説明］	宣言されたビット・フィールドの幅はビット・フィールドの型より大きいです。any-string ビットに縮小されます。
W0520961	［メッセージ］	Use of a type with no linkage to declare a variable with linkage.
	［説明］	リンケージを持たない型がリンケージを持つ変数宣言に使用されました。
W0520962	［メッセージ］	Use of a type with no linkage to declare a function.
	［説明］	リンケージを持たない型が関数に使用されました。
W0520970	［メッセージ］	The qualifier on this friend declaration is ignored.
	［説明］	このフレンド宣言への修飾子は無視されます。
W0520973	［メッセージ］	Inline used as a function qualifier is ignored.
	［説明］	inline 関数修飾子は無視されました。
W0520984	［メッセージ］	operator new and operator delete cannot be given internal linkage.
	［説明］	operator new/operator delete が static で定義されています。
W0520991	［メッセージ］	Extra braces are nonstandard.
	［説明］	余分な括弧は標準ではありません。
W0520993	［メッセージ］	Subtraction of pointer types " *型名 1*" and " *型名 2*" is nonstandard.
	［説明］	" *型名 1*" と " *型名 2*" のポインタ型の減算は標準ではありません。
W0520997	［メッセージ］	*関数名 2* is hidden by *関数名 1* -- virtual function override intended?
	［説明］	" *関数名 1*" が " *関数名 2*" を隠しています。仮想関数をオーバーライドしようとしていないか確認してください。
W0521000	［メッセージ］	A storage class may not be specified here.
	［説明］	記憶域クラスはここでは指定できません。
W0521028	［メッセージ］	Invalid redeclaration of nested class.
	［説明］	クラス内でクラスを二重定義しています。
W0521030	［メッセージ］	A variable with static storage duration cannot be defined within an inline function.
	［説明］	静的変数は inline 関数で定義できません。
W0521046	［メッセージ］	Floating-point value cannot be represented exactly.
	［説明］	浮動小数点の値が正しく記述されていません。

W0521050	[メッセージ]	Imaginary *= imaginary sets the left-hand operand to zero.
	[説明]	虚数 *= 虚数は左オペランドを 0 にします。
W0521051	[メッセージ]	Standard requires that "*symbol*" be given a type by a subsequent declaration ("int" assumed).
	[説明]	*symbol* の型は直後の宣言により与えられなければなりません ("int" に仮定されます)。
W0521053	[メッセージ]	Conversion from integer to smaller pointer.
	[説明]	整数がそれより幅の小さなポインタに変換されました。
W0521055	[メッセージ]	Types cannot be declared in anonymous unions.
	[説明]	型を無名共用体内で宣言することはできません。
W0521056	[メッセージ]	Returning pointer to local variable.
	[説明]	ローカル変数へのポインタが返却されました。
W0521057	[メッセージ]	Returning pointer to local temporary.
	[説明]	ローカルな領域へのポインタを返しています。
W0521072	[メッセージ]	A declaration cannot have a label.
	[説明]	宣言はラベルを持てません。
W0521105	[メッセージ]	#warning directive: *文字列* .
	[説明]	#warning 指令 : *文字列*。
W0521145	[メッセージ]	*型 1* would have been promoted to " *型 2*" when passed through the ellipsis parameter; use the latter type instead.
	[説明]	*型 1* は *型 2* へと拡張されます。*型 2* を使用します。
W0521163	[メッセージ]	va_start should only appear in a function with an ellipsis parameter.
	[説明]	va_start が使用されるのは省略記号を引数とする関数のみです。
W0521192	[メッセージ]	Null (zero) character in input line ignored.
	[説明]	null 文字 (zero) の入力は無視されました。
W0521193	[メッセージ]	Null (zero) character in string or character constant.
	[説明]	null 文字 (zero) が文字列か文字定数にありました。
W0521194	[メッセージ]	Null (zero) character in header name.
	[説明]	null 文字 (zero) がヘッダ・ファイル名にあります。
W0521197	[メッセージ]	The prototype declaration of %nfd is ignored after this unprototyped redeclaration.
	[説明]	関数原型を無視します。
W0521211	[メッセージ]	Nonstandard cast to array type ignored.
	[説明]	標準でない配列型へのキャストは無視されました。
W0521213	[メッセージ]	field uses tail padding of a base class.
	[説明]	フィールドは基底クラスの終端パディングを使用しています。
W0521218	[メッセージ]	Base class *xxx* uses tail padding of base class *yyy*.
	[説明]	基底クラス 1 は基底クラス 2 の終端パディングを使用しています。

W0521222	[メッセージ]	Invalid error number.
	[説明]	不正なエラー番号です。
W0521223	[メッセージ]	Invalid error tag.
	[説明]	不正なエラータグです。
W0521224	[メッセージ]	Expected an error number or error tag.
	[説明]	エラー番号かエラータグがありません。
W0521235	[メッセージ]	Nonstandard conversion between pointer to function and pointer to data.
	[説明]	非標準形式の変換がポインタ関数と不完全なオブジェクト間で行われました。
W0521273	[メッセージ]	Alignment-of operator applied to incomplete type.
	[説明]	オペレータのアライメントが不完全な型に対して適用されました。
W0521285	[メッセージ]	Nonstandard qualified name in namespace member declaration.
	[説明]	非標準形式の修飾子名が名前空間のメンバの宣言に使用されています。
W0521290	[メッセージ]	Non-POD class type passed through ellipsis.
	[説明]	非 POD クラス型が省略記号に渡されています。
W0521294	[メッセージ]	Integer operand may cause fixed-point overflow.
	[説明]	整数オペランドは固定小数点でオーバフローを起こす原因となります。
W0521296	[メッセージ]	Fixed-point value cannot be represented exactly.
	[説明]	固定小数点値が正確に表現できません。
W0521297	[メッセージ]	Constant is too large for long long; given unsigned long long type (nonstandard).
	[説明]	定数が long long 型には大きすぎます。unsigned long long 型にしてください（標準ではありません）。
W0521301	[メッセージ]	*xxx* declares a non-template function -- add <> to refer to a template instance.
	[説明]	非テンプレート関数を宣言しています。
W0521302	[メッセージ]	Operation may cause fixed-point overflow.
	[説明]	処理は固定小数点のオーバフローを起こす原因となります。
W0521307	[メッセージ]	Class member typedef may not be redeclared.
	[説明]	クラスメンバの typedef を再宣言してはいけません。
W0521308	[メッセージ]	Taking the address of a temporary.
	[説明]	テンポラリのアドレスが取得されました。
W0521310	[メッセージ]	Fixed-point value implicitly converted to floating-point type.
	[説明]	固定小数点値が暗黙に浮動小数点型に変換されました。
W0521316	[メッセージ]	Value cannot be converted to fixed-point value exactly.
	[説明]	値は厳密に固定小数点値に変換できません。
W0521319	[メッセージ]	Fixed-point operation result is out of range.
	[説明]	固定小数点の処理が範囲を越えました。
W0521342	[メッセージ]	const_cast to enum type is nonstandard.
	[説明]	const_cast で列挙型をキャストするのは標準形式ではありません。

W0521346	［メッセージ］	Function returns incomplete class type %t.
	［説明］	関数が不正なクラス型を返しています。
W0521361	［メッセージ］	Negation of an unsigned fixed-point value.
	［説明］	符号なし固定小数点値を反転しました。
W0521373	［メッセージ］	Implicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
	［説明］	64 ビット整数型がより小さい整数型へと暗黙的に変換されています。移植性の問題になる可能性があります。
W0521374	［メッセージ］	Explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem).
	［説明］	64 ビット整数型がより小さい整数型へと明示的に変換されています。移植性の問題になる可能性があります。
W0521375	［メッセージ］	Conversion from pointer to same-sized integral type (potential portability problem).
	［説明］	ポインタが同じサイズの整数型に変換されました（潜在的な移植性の問題があります）。
W0521386	［メッセージ］	Storage specifier ignored.
	［説明］	記憶クラス指定子を無視します。
W0521396	［メッセージ］	White space between backslash and newline in line splice ignored.
	［説明］	連結行のバック・スラッシュと改行の間に空白類がありました。無視します。
W0521400	［メッセージ］	positional format specifier cannot be zero.
	［説明］	positional format 指定子は 0 を指定できません。
W0521420	［メッセージ］	Some enumerator values cannot be represented by the integral type underlying the enum type.
	［説明］	いくつかの列挙子はその列挙型の潜在的な整数型で表現できません。
W0521422	［メッセージ］	Multicharacter character literal (potential portability problem).
	［説明］	多バイト文字リテラルです。潜在的な移植性の問題があります。
W0521427	［メッセージ］	offsetof applied to non-POD types is nonstandard.
	［説明］	offsetof を non-POD 型に指定するのは標準ではありません。
W0521433	［メッセージ］	No prior push_macro for *xxx*.
	［説明］	*xxx* の push_macro は優先されません。
W0521443	［メッセージ］	__real/__imag applied to real value.
	［説明］	__real/__imag が実数に指定されました。
W0521444	［メッセージ］	*symbol* was declared "deprecated (*xxx*)".
	［説明］	*symbol* は "deprecated (*xxx*)" に宣言されました。
W0521546	［メッセージ］	Argument must be a constant null pointer value.
	［説明］	引数は null ポインタ値でなければなりません。
W0521547	［メッセージ］	Insufficient number of arguments for sentinel value.
	［説明］	センチネル値への引数番号は不適当です。

W0521548	［メッセージ］	Sentinel argument must correspond to an ellipsis parameter.
	［説明］	センチネル引数は省略引数に一致する必要があります。
W0521551	［メッセージ］	No #pragma start_map_region is currently active: pragma ignored.
	［説明］	#pragma start_map_region は現在有効ではありません : pragma は無視されました。
W0521553	［メッセージ］	Nonstandard empty wide character literal treated as L'0'.
	［説明］	非標準の空のワイド文字リテラルは L'0' として扱います。
W0521561	［メッセージ］	Predefined meaning of "*symbol*" discarded.
	［説明］	すでに宣言された "*symbol*" の意味はなくなりました。
W0521564	［メッセージ］	enum qualified name is nonstandard.
	［説明］	enum 指定子は標準ではありません。
W0521565	［メッセージ］	Anonymous union qualifier is nonstandard.
	［説明］	無名共用体の指定子は標準ではありません。
W0521566	［メッセージ］	Anonymous union qualifier is ignored.
	［説明］	無名共用体の指定子は無視されました。
W0521570	［メッセージ］	Nonstandard specifier ignored.
	［説明］	標準でない指定子は無視されました。
W0521607	［メッセージ］	#pragma text must precede the definition of function "*function*".
	［説明］	関数 " *関数名* " はすでに定義されています。#pragma text を指定できません。指定を無視します。
W0521644	［メッセージ］	Definition at end of file not followed by a semicolon or a declarator.
	［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］	White space is required between the macro name "*macro name*" and its replacement text
	［対処方法］	マクロ名とその置換テキストの間に空白を入れて区切ってください。
W0523042	［メッセージ］	Using *xxx* function at influence the code generation of "SuperH" compiler.
	［説明］	SuperH コンパイラとの互換性に影響する可能性があります。仕様相違の詳細をご確認ください。
W0523060	［メッセージ］	Incompatible section specified.
	［説明］	同一識別子が複数回宣言されており、宣言ごとに異なるセクションが指定されています。最初のセクションを有効にします。
W0523063	［メッセージ］	" *文字列* " has no effect in this version.
	［説明］	このバージョンでは " *文字列* " は無効です。
W0523064	［メッセージ］	Address taken " *変数名* ". It may cause an upset endian indirect reference.
	［説明］	endian オプションと異なるエンディアンの 8 バイト変数 " *変数名* " のアドレスが取得されました。エンディアン処理が正しくない間接参照を引き起こす可能性があります。
W0530809	［メッセージ］	const qualifier for variable " *変数名* " differs between files.
	［説明］	" *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］	Type of symbol " *シンボル名* " differs between files.
	［説明］	" *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。
W0544001	［メッセージ］	Alignment of " *セクション名* " sections is inconsistent. " *数値* " is assumed.
	［説明］	セクションの命名に誤りがあります。アライメント数の異なるセクションに同じ名前が付いています。
W0544002	［メッセージ］	Endian of " *セクション名* " sections is inconsistent. " *エンディアン種別* " is assumed.
	［説明］	セクションの命名に誤りがあります。エンディアンの異なるセクションに同じ名前が付いています。
W0551000	［メッセージ］	'.ALIGN' with not 'ALIGN' specified relocatable section.
	［説明］	ALIGN 指定がないセクション内に制御命令 ".ALIGN" が記述されています。
	［対処方法］	制御命令 ".ALIGN" の記述位置を確認してください。制御命令 ".ALIGN" を記述するセクションのセクション定義行に ALIGN 指定を記述してください。
W0551001	［メッセージ］	Destination address may be changed.
	［説明］	分岐先が期待するものと異なる位置になる可能性があります。
	［対処方法］	アドレッシングモードが最適選択されないように分岐命令のオペランドを記述してください。
W0551002	［メッセージ］	Floating point value is out of range.
	［説明］	浮動小数点数が範囲外です。
	［対処方法］	浮動小数点数の記述を確認してください。範囲外は無視します。
W0551003	［メッセージ］	Location counter exceed.
	［説明］	ロケーションカウンタが 0FFFFFFFFh を越えました。
	［対処方法］	.ORG のオペランド値を確認してください。ソースを記述し直してください。
W0551004	［メッセージ］	'.ALIGN' size is different.
	［説明］	アライメント値が異なります。
	［対処方法］	アライメント値を確認してください。
W0551006	［メッセージ］	Data in 'CODE' section align in 4byte.
	［説明］	endian=big 時、CODE セクション中データ領域の開始位置は 4 バイト境界に補正されます。
W0551007	［メッセージ］	Data size in 'CODE' section align in 4byte.
	［説明］	endian=big 時、CODE セクション中データ領域のサイズは 4 の倍数に補正されます。
W0551009	［メッセージ］	Multiple symbols.
	［説明］	.STACK でシンボルへのスタック値指定を重複して行なっています。
W0551010	［メッセージ］	Section attribute mismatch.
	［説明］	セクションの属性が異なっています。
W0551011	［メッセージ］	Use PM instruction.
	［説明］	特権命令を使用しています。

W0551012	［メッセージ］	Use FPU instruction.
	［説明］	浮動小数点演算命令を使用しています。
W0551013	［メッセージ］	Use DSP instruction.
	［説明］	DSP 機能命令を使用しています。
W0551014	［メッセージ］	Too many actual macro parameters.
	［説明］	マクロ実引数の数が多すぎます。余分な実引数は無視されます。
W0551015	［メッセージ］	Actual macro parameters are not enough.
	［説明］	マクロ実引数の数がマクロ仮引数の数より少なくなっています。該当する実引数のない仮引数は無効となります。
W0551016	［メッセージ］	'.END' statement is in include file.
	［説明］	インクルード・ファイルに .END 記述があります。.END を無視して処理します。
	［対処方法］	インクルード・ファイル内には、.END は記述できません。記述を削除してください。
W0551017	［メッセージ］	Use of < 命令種別 > instruction found.
	［説明］	< 命令種別 > で示す命令を使用しています。
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］	Fatal レベル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。
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。
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 (128Kbyte) 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。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。
W0561100	［メッセージ］	Cannot find "*名前*" specified in option "*オプション*"
	［説明］	"*オプション*" で指定したシンボル名、またはセクション名が見つかりません。"*名前*" の指定を無視します。
W0561101	［メッセージ］	"*名前*" in rename option conflicts between symbol and section
	［説明］	rename オプションで指定した "*名前*" がセクション名とシンボル名の両方に存在します。シンボル名を変更の対象にします。
W0561102	［メッセージ］	Symbol "*シンボル*" redefined in option "*オプション*"
	［説明］	"*オプション*" で指定したシンボルはすでに定義されています。そのまま処理を続けます。
W0561103	［メッセージ］	Invalid address value specified in option "*オプション*" : "*アドレス*"
	［説明］	"*オプション*" で指定した "*アドレス*" は無効な値です。"*アドレス*" の指定を無視します。
W0561104	［メッセージ］	Invalid section specified in option "*オプション*" : "*セクション*"
	［説明］	"*オプション*" で無効なセクションを指定しています。
	［対処方法］	以下を確認してください。 (1) -output オプションは、初期値のないセクションを指定できません。 (2) -jump_entries_for_pic オプションは、プログラムセクション以外を指定できません。
W0561110	［メッセージ］	Entry symbol "*シンボル*" in entry option conflicts
	［説明］	entry オプションで指定した "*シンボル*" 以外のシンボルがコンパイル、アセンブル時にエントリ・シンボルとして指定されています。オプション指定を優先します。

W0561120	［メッセージ］	Section address is not assigned to " *セクション* "
	［説明］	" *セクション* " のアドレス指定がありません。" *セクション* " を最後尾に配置します。
	［対処方法］	rlink オプション -start を使用して、セクションのアドレスを設定してください。
W0561121	［メッセージ］	Address cannot be assigned to absolute section " *セクション* " in start option
	［説明］	" *セクション* " は絶対アドレス・セクションです。絶対アドレス・セクションに対するアドレス指定を無視します。
W0561122	［メッセージ］	Section address in start option is incompatible with alignment : " *セクション* "
	［説明］	start オプションで指定した " *セクション* " のアドレスはアライメント数と矛盾しています。アライメント数に合わせてセクションアドレスを補正します。
W0561130	［メッセージ］	Section attribute mismatch in rom option : " *セクション1* "," *セクション2* "
	［説明］	rom オプションで指定した " *セクション1* " と " *セクション2* " の属性、アライメント数が異なります。" *セクション2* " のアライメント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］	-create_unfilled_area オプションで生成した " *シンボル* " が一対になっていません。
W0561143	［メッセージ］	Address range *開始アドレス* - *終了アドレス* cannot be placed in 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ありません。そのため、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。リンカは、" *セクション* " を自動生成しません。
W0561183	［メッセージ］	Interrupt number " *ベクタ番号* " of " *セクション* " is defined in input file
	［説明］	VECTN オプションで記述したベクタ番号は、入力ファイル内で定義済みです。入力ファイルの内容を優先して、処理を継続します。

W0561184	［メッセージ］	Interrupt number " *ベクタ番号* " of " *セクション* " is defined
	［説明］	debug_monitor で使用するベクタ番号は、入力ファイル内または vectn で定義済みです。 debug_monitor の指定を優先して、処理を継続します。
W0561190	［メッセージ］	Section " *セクション* " was moved other area specified in option "*cpu*=< *メモリ属性* >"
	［説明］	外部変数アクセス最適化によりオブジェクト・サイズが変更されたため、次の cpu 指定範囲の " *セクション* " を移動しました。
W0561191	［メッセージ］	Area of "FIX" is within the range of the area specified by "*cpu*=< *メモリ属性* >" :"<*start*>-<*end*>"
	［説明］	cpu オプションで、メモリ属性 FIX と FIX 以外の <*start*>-<*end*> 範囲が重複していたため、FIX を有効にしました。
W0561192	［メッセージ］	Bss Section " *セクション名* " is not initialized
	［説明］	初期値なしのデータ・セクション " *セクション名* " は、初期設定プログラムで初期化できません。
	［対処方法］	-cpu 指定範囲、ポインタ変数のサイズ指定を見直してください。
W0561193	［メッセージ］	Section " *セクション名* " specified in option " *オプション* " is ignored
	［説明］	-cpu=stride の機能で分割したセクションの、後半部への " *オプション* " 指定は無効となります。
	［対処方法］	後半部のセクションは " *オプション* " で指定しないでください。
W0561194	［メッセージ］	Section " *セクション* " in relocation " *ファイル* "-" *セクション* "-" *オフセット* " is changed.
	［説明］	" *セクション* "" *ファイル* "" *オフセット* "　の位置にある " *セクション* " を参照していたリロケーションが、分割した後半セクションを参照するよう変更しました。
	［対処方法］	分割しないようにするには、" *セクション* " を contiguous_section オプションで指定してください。
W0561200	［メッセージ］	Backed up file " *ファイル 1* " into " *ファイル 2* "
	［説明］	入力ファイル " *ファイル 1* " は書き換えられました。書き換える前の " *ファイル 1* " の内容は " *ファイル 2* " にバックアップされています。
W0561300	［メッセージ］	Option " *オプション* " is ineffective without debug information
	［説明］	入力ファイル内にデバッグ情報がありません。" *オプション* " 指定を無視します。
	［対処方法］	コンパイル、アセンブル時に該当するオプションを指定しているか確認してください。
W0561301	［メッセージ］	No inter-module optimization information in input files
	［説明］	入力ファイル内にモジュール間最適化情報がありません。optimize オプションを無視します。
	［対処方法］	コンパイル、アセンブル時に goptimize オプションを指定してください。
W0561302	［メッセージ］	No stack information in input files
	［説明］	入力ファイル内にスタック情報がありません。stack オプションを無視します。入力ファイルがアセンブラ出力ファイルの場合は、stack オプションは無効です。
W0561305	［メッセージ］	Entry address in " *ファイル* " conflicts : " *アドレス* "
	［説明］	異なるエントリーアドレスのファイルが複数入力されています。

W0561310	［メッセージ］	" *セクション*" in " *ファイル*" is not supported in this tool
	［説明］	" *ファイル*" 内に非サポートセクションがありました。" *セクション*" を無視します。
W0561311	［メッセージ］	Invalid debug information format in " *ファイル*"
	［説明］	" *ファイル*" 内のデバッグ情報は dwarf2 ではありません。debug 情報を削除します。
W0561320	［メッセージ］	Duplicate symbol " *シンボル*" in " *ファイル*"
	［説明］	" *シンボル*" は重複しています。先に入力したファイル内シンボルを優先します。
W0561321	［メッセージ］	Entry symbol " *シンボル*" in " *ファイル*" conflicts
	［説明］	エントリ・シンボル定義のあるオブジェクト・ファイルを複数入力しました。先に入力したファイル内のエントリ・シンボルを有効にします。
W0561322	［メッセージ］	Section alignment mismatch : " *セクション*"
	［説明］	アライメント数の異なる同名セクションを入力しました。アライメント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、他のファイルの同名シンボルと属性が一致していません。
	［対処方法］	シンボルを確認してください。
W0561326	［メッセージ］	Reserved symbol " *シンボル*"is defined in " *ファイル*"
	［説明］	予約された名称のシンボル " *シンボル*" が " *ファイル*" 内で定義されています。
W0561327	［メッセージ］	Section alignment in option "aligned_section" is small : " *セクション*"
	［説明］	aligned_section オプション指定時のアライメント数 16 の方が、" *セクション*" のアライメント数より小さいため、指定セクションに対するオプション指定を無視します。
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。

W0561430	［メッセージ］	Cannot generate effective bls file for compiler optimization
	［説明］	無効な bls ファイルが生成されました。コンパイル時に、外部変数アクセス最適化 (map オプション) を指定しても、この最適化は実施できません。
	［対処方法］	コンパイラの外部変数アクセス最適化 (map オプション) には、以下の制限があります。該当する内容がないかを確認し、セクション配置を見直してください。 コンパイル時に base オプションを使用している場合、プログラムセクションの直後にデータ・セクションを配置すると、外部変数アクセス最適化が実施できない場合があります。 なお、bls ファイルは “ 外部シンボル割り付け情報ファイル ” を指します。これは、コンパイラの map オプションに使用するための情報ファイルです。
W0561510	［メッセージ］	Input file was compiled with option "smap" and option "map" is specified at linkage
	［説明］	smap を指定してコンパイルしたファイルがあります。
	［対処方法］	smap を指定したファイルは、2 回目のビルドで map オプションを指定してコンパイルしないでください。
W0571600	［メッセージ］	An error occurred during name decoding of " *インスタンス* "
	［説明］	" *インスタンス* " がデコードできませんでした。エンコード名でメッセージ出力します。
W0578306	［メッセージ］	can not open file *file*
	［説明］	ファイル *file* をオープンできません。
W0578307	［メッセージ］	can not close file *file*
	［説明］	ファイル *file* をクローズできません。
W0578308	［メッセージ］	can not read file *file*
	［説明］	ファイル *file* からの読み込みができません。
W0578309	［メッセージ］	can not write file *file*
	［説明］	ファイル *file* への書き込みができません。
W0578310	［メッセージ］	can not seek file *file*
	［説明］	ファイル *file* をシークできません。
W0578311	［メッセージ］	can not find file *file*
	［説明］	ファイル *file* を見つけることができません。
W0578315	［メッセージ］	can not find member *member*
	［説明］	ライブラリ・ファイル内にメンバ *member* が存在しません。
W0578322	［メッセージ］	this symbol offset not true
	［説明］	ライブラリ・ファイル内のシンボルオフセットが不正です。
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。

10.5.6 C 標準ライブラリ関数のエラーメッセージ

ライブラリ関数の中には、ライブラリ関数を実行中にエラーが発生した場合、標準ライブラリのヘッダファイル <errno.h> で定義しているマクロ errno にエラー番号を設定するものがあります。

エラー番号には対応するエラーメッセージが定義してあり、エラーメッセージを出力することができます。エラーメッセージを出力するプログラム例を以下に示します。

例

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

main()
{
    FILE *fp;

    fp=fopen("file", "w");
    fp=NULL;

    fclose(fp);                       /* error occurred */

    printf("%s¥n", strerror(errno));  /* print error message */
}
```

説明

(1) fclose 関数に値 NULL のファイルポインタを実引数として渡しているので、エラーとなります。このとき errno に対応するエラー番号が設定されます。

(2) strerror 関数は、エラー番号を実引数として渡すと、対応するエラーメッセージの文字列のポインタを返します。printf 関数の文字列出力指定によりエラーメッセージを出力します。

標準ライブラリエラーメッセージ一覧

エラー番号	エラーメッセージ／説明	エラー番号を設定する関数
0x22 (ERANGE)	Data out of range オーバーフローが発生しました。	frexp, ldexp, modf, ceil, floor, fmod, atof, atoi, atol, atoll, atolfixed, atolaccum, strtod, strtol, strtoul, strtoll, strtoull, strtolfixed, strtolaccum, perror, fprintf, fscanf, printf, scanf, sprintf, sscanf, vfprintf, vprintf, vsprintf, acos, acosf, asin, asinf, atan, atan2, atan2f, atanf, ceilf, cos, cosf, cosh, coshf, exp, expf, floorf, fmodf, ldexpf, log,log10, log10f, logf, modff, pow, powf, sin, sinf, sinh, sinhf, sqrt, sqrtf, tan, tanf, tanh, tanhf, fabs, fabsf, frexpf
0x21 (EDOM)	Data out of domain 数学関数の引数に対する結果の値が定義されません。	acos, acosf, asin, asinf, atan, atan2, atan2f, atanf, ceil, ceilf, cos, cosf, cosh, coshf, exp, expf, floor, floorf, fmod, fmodf, ldexp, ldexpf, log, log10, log10f, logf, modf, modff, pow, powf, sin, sinf, sinh, sinhf, sqrt, sqrtf, tan, tanf, tanh, tanhf, fabs, fabsf, frexp, frexpf
0x450 (ESTRN)	Too long string 文字列の文字数が 512 文字を超えています。	atof, atoi, atol, atoll, atolfixed, atolaccum, strtod, strtol, strtoul, strtoll, strtoull, strtolfixed, strtolaccum
0x04B0 (ECBASE)	Invalid radix 基数の指定が誤っています。	strtol, strtoul, strtoll, strtoull
0x04B2 (ETLN)	Number too long 数値を表現する文字列の文字数が有効桁数を超えています。	atof, atolfixed, atolaccum, strtod, strtolfixed, strtolaccum, fscanf, scanf, sscanf
0x04B4 (EEXP)	Exponent too large 指数部の桁数が 3 桁を超えています。	atof, strtod, fscanf, scanf, sscanf
0x04B6 (EEXPN)	Normalized exponent too large 文字列を一度 IEEE 規格の 10 進形式に正規化したとき指数部の桁数が 3 桁を超えています。	atof, strtod, fscanf, scanf, sscanf
0x04BA (EFLOATO)	Overflow out of float float 型の 10 進数値が、float 型の範囲を超えています（オーバーフロー）。	fscanf, scanf, sscanf
0x04C4 (EFLOATU)	Underflow out of float float 型の 10 進数値が、float 型の範囲を超えています（アンダーフロー）。	fscanf, scanf, sscanf
0x04E2 (EDBLO)	Overflow out of double double 型の 10 進数値が、double 型の範囲を超えています（オーバーフロー）。	fscanf, scanf, sscanf
0x04EC (EDBLU)	Underflow out of double double 型の 10 進数値が、double 型の範囲を超えています（アンダーフロー）。	fscanf, scanf, sscanf
0x04F6 (ELDBLO)	Overflow out of long double long double 型の 10 進数値が、long double 型の範囲を超えています（オーバーフロー）。	fscanf, scanf, sscanf
0x0500 (ELDBLU)	Underflow out of long double long double 型の 10 進数値が、long double 型の範囲を超えています（アンダーフロー）。	fscanf, scanf, sscanf

11. 注意事項

この章では、CC-RX を用いる際に注意すべき点について説明します。

11.1 コーディング上の注意事項

(1) 関数原型について
関数を呼び出す際には、呼び出される関数の関数原型を行ってください。関数原型を行わない場合、パラメータの受け渡しが正しく行えない場合があります。

例 1. float 型パラメータをもつ関数（dbl_size=8 を指定した場合）

```
void g()
{
            float a;
            ...
            f(a);                       // a は double 型に変換されます。
}
void f(float x)
{...}
```

例 2. スタック渡しとなる signed char、(unsigned)char、(signed)short、および unsigned short 型のパラメータをもつ関数

```
void h();
void g()
{
            char a,b;
            ...
            h(1,2,3,4,a,b);             // a,b は int 型に変換されます。
}
void h(int a1, int a2, int a3, int a4, char a5, char a6)
{...}
```

(2) 引数に型情報のない関数宣言
同じ関数に対して関数宣言 (関数定義を含む) を複数行うとき、引数並びに型を記述しない形式と、型を記述する形式を両方使用しないでください。
この場合、呼び出す関数と呼び出される関数とで引数の解釈に違いが生じるため、生成コードが型を正しく処理できない場合があります。
コンパイル時に W0520147 のメッセージが表示された場合、この問題に該当している可能性がありますので、引数並びに型を記述する形式に変更するか、生成コードを確認して引数の受け渡しに問題がないかを確認してください。

例
old_style を異なる形式で記述しているために、引数 d と e の型の意味が呼び出す関数と呼び出される関数で異なるため、引数の受け渡しが正しく行われません。

```
extern int old_style(int,int,int,short,short);  /* 関数宣言：引数並びに型を記述する形式
*/
int old_style(a,b,c,d,e)    /* 関数定義： 引数並びに型を記述しない形式 */
 int a,b,c;
 short d,e;
{
    return a + b + c + d + e;
}
int result;
func()
{
    result = old_style(1,2,3,4,5);
}
```

(3) **C/C++** 言語で評価順序を規定していない式

C/C++ 言語規格で評価順序が規定されていない式を用いる際、評価順序で結果が変わるようなコーディングをした場合は動作保証しません。

例

```
a[i]=a[++i];  代入式の右辺を先に評価するか後に評価するかで左辺の値が変わります。

sub(++i, i);  関数の第１引数を先に評価するか後に評価するかで第２引数の値が変わります。
```

(4) オーバーフロー演算、ゼロ除算

オーバーフロー演算や浮動小数点のゼロ除算があっても、エラーメッセージを出力しません。ただし、一つの定数または定数どうしの演算でのオーバーフロー演算があれば、コンパイル時にエラーメッセージを出力します。

例

```
void main()
{
  int ia;
  int ib;
  float fa;
  float fb;

  ib=32767;
  fb=3.4e+38f;

  /* 定数または定数どうしの演算時はオーバーフローに対する                    */
  /* コンパイルエラーメッセージを出力します                                  */

  ia=99999999999;          /* (W) 定数のオーバーフローを検出します          */
  fa=3.5e+40f;             /* (E) 浮動小数点演算のオーバーフローを検出します */

  /* 実行時のオーバーフローに対するエラーメッセージは出力しません           */

  ib=ib+32767;             /* 演算結果のオーバーフローを無視します          */
  fb=fb+3.4e+38f;          /* 浮動小数点演算結果のオーバーフローを無視します */
}
```

(5) **const** 型変数への書き込み

const 型の変数を宣言していても、型変換で const 型でない型に変換して代入した場合や、分割コンパイルしたプログラムの間で、型を統一して扱っていない場合は、const 型変数への書き込みをコンパイラでチェックできませんので、注意が必要です。

例

```
const char *p;                    /* ライブラリ関数 strcat の第１引数は char 型への */
  :                               /* ポインタ型なので、引数の指す領域が書き換わる   */
strcat(p, "abc");                 /* ことがあります                                 */

  ファイル１
const int i;

  ファイル２
extern int i;                     /* 変数 i は、ファイル２では const 型で宣言していま */
  :                               /* せんのでファイル２の中で書き込んでもエラーに   */
i=10;                             /* なりません                                     */
```

(6) 数学関数ライブラリの精度について

acos(x)、asin(x) 関数では x ≒ 1 で誤差が大きくなりますので注意が必要です。
誤差範囲は以下のとおりです。

acos(1.0 -ε) における絶対誤差	倍精度 2^{-39} (ε= 2^{-33})
	単精度 2^{-21} (ε= 2^{-19})
asin(1.0 -ε) における絶対誤差	倍精度 2^{-39} (ε= 2^{-28})
	単精度 2^{-21} (ε= 2^{-16})

(7) 最適化により削除される可能性のあるコーディング

連続した同一変数の参照や、結果を使用しない式を記述した場合、コンパイラの最適化により冗長コードとして削除される場合があります。常にアクセスを保証する場合は、宣言時に volatile を指定してください。

例

```
[1] b=a;              /* １行目の式は冗長コードとして削除されることがあります     */
    b=a;
[2] while(1)a;        /* 変数 a の参照およびループ文は冗長コードとして削除される  */
                      /* ことがあります                                           */
```

(8) **C89** と **C99** の動作の差異

C99 では、選択文および反復文は、{} で囲まれます。そのため、C89 と C99 で動作が異なることがあります。

例

```
enum {a,b};
int g(void)
{
     if(!sizeof(enum{b,a}))
     return a;
     return b;
}
```

上記を -lang=c99 を指定してコンパイルすると、以下の解釈となります。

```
enum {a,b};
int g(void)
{
     {
         if(!sizeof(enum{b,a}))
         return a;
     }
     return b;
}
```

-lang=c では、g()=0 となりますが、-lang=c99 では、g()=1 となります。

(9) オーバーフローを伴う演算および型変換に関する注意事項
数値演算や型変換を行う場合、その型で取り扱える値の範囲外 (オーバーフロー) とならないようにご注意ください。オーバーフローが発生すると、得られる結果がコンパイルオプションなどの条件によって変化する場合があります。
標準の C 言語では、オーバーフローを伴う演算処理の結果は未定義となっており、コンパイル条件などにより得られる結果が異なる場合があります。
演算処理を行うプログラムでは、オーバーフローを発生させないようにご注意ください。
実際に結果が異なる例を次に示します。

例 float 型→ unsigned short 型への変換

```
float f = 2147483648.0f;
unsigned short ui2;
void ex1func(void)
{
        ui2 = f; /* float → unsigned short への変換 */
}
```

ex1func 関数を実行して得られる ui2 の値は、FPU あり (-fpu) と FPU なし (-nofpu) とで次のように異なります。
FPU あり : ui2 = 65535
FPU なし : ui2 = 0
これは、float 型から unsigned short 型への変換の方法が FPU ありと FPU なしで異なるためです。

(10) アンダーバー(_) を 2 つ並べたシンボルの記述
アンダーバーを 2 つ以上並べたシンボルは記述しないようにしてください。
生成されるコードは問題なく動作しますが、リンクマップ表示などでは関数名がそのまま表示されず、異なる名前の C++ の関数名として表示される場合があります。

例

```
int sample__Fc(void) { return 0; }
```

この場合、リンクマップ出力には、_sample__Fc ではなく、sample(char) と表示されます。

11.2 C プログラムを C++ コンパイラでコンパイルするときの注意事項

(1) 関数原型
関数を使用する前に関数原型が必要です。そのときには、仮引数の型も必ず宣言してください。

```
extern void func1();
void g()
{
    func1(1); // エラー
}
```

```
extern void func1(int);
void g()
{
    func1(1); // OK
}
```

(2) **const** オブジェクトのリンケージ
const オブジェクトのリンケージは、C プログラムでは外部結合であるのに対し、C++ プログラムでは内部結合になります。また、const オブジェクトは初期値を必要とします。

```
const cvalue1;     // エラー

const cvalue2 = 1; // 内部結合
```

```
const cvalue1 = 0;         // 初期値を与えます

extern const cvalue2 = 1; // C プログラムと同様に外部結合になります
```

(3) **void*** からの代入
C++ プログラムでは、明示的なキャストを用いないと他のオブジェクト型へのポインタ (関数へのポインタ、メンバへのポインタを除く) へ代入できません。

```
void func(void *ptrv, int *ptri)
{
    ptri = ptrv; // エラー
}
```

```
void func(void *ptrv, int *ptri)
{
    ptri = (int *)ptrv; // OK
}
```

11.3 オプションに関する注意事項

(1) 指定の統一が必要なオプションについて
指定の統一が必要なオプションを以下 (a) (b) に示します。これらのオプション指定の異なるリロケータブルファイルおよびライブラリファイルをリンクした場合、実行時の動作は保証しません。

(a) (a) isa、cpu、endian、base、および fint_register の 5 つのオプションは、コンパイラ、アセンブラ、およびライブラリジェネレータで統一してください。

(b) (b) 「コマンド・リファレンス」の章の「マイコンオプション」に該当する上記 (a) 以外のオプションについては、コンパイラおよびライブラリジェネレータで統一してください。

(2) lbgrx(ライブラリジェネレータ) の -reent(リエントラントライブラリを生成するオプション) をご利用の場合
ルネサス統合開発環境で生成されたプロジェクト上で、ライブラリジェネレータ lbgrx に対し、-reent オプションの指定を有効にする場合、プロジェクト内の低水準関数の _INIT_IOLIB() 関数に、配列 _Files に対する次の実行文が含まれるかをご確認ください。
もし記述がない場合は、「8.4 コーディング例」の「(e) lowsrc.c -- 低水準インタフェースルーチン (C 言語部分)」の _INIT_IOLIB() 関数の内容を参考に、これらの実行文を追加してください。

```
_Files[0] = stdin;
_Files[1] = stdout;
_Files[2] = stderr;
```

11.4 最適化リンケージエディタにおいて最適化有効時の E0562330 エラー発生の回避

最適化リンケージエディタは、命令の参照先シンボルの配置アドレスに応じてコードサイズの小さな命令に置き換える最適化を行います。よりコードサイズを小さくする効果を得るために、より多くの命令に対して置き換えを試みます。

しかし、命令の置き換えを実施した結果、参照先のシンボルの配置アドレスが変わってしまい、置き換えられた命令ではシンボルを参照できなくなってしまうケースがあります。最適化リンケージエディタでは、このような場合に問題のあるコードを生成しないために E0562330 エラーを出力して停止します。

この事象は、RX アーキテクチャの特性から、最適化前は FFFF8000h 番地以上にあるシンボル（変数や定数、スイッチテーブルなど）が最適化後に FFFF8000h 番地以下に配置を変更された場合に発生することがあります。

(1) 概略
以下にエラー発生に至るまでの概略を示します。

定数 CONST1,CONST2 を読み出すプログラムにおいて、最適化前の P セクションと C セクションが次のようにあり、P セクションに後続して C セクションが FFFF8000h に配置されていたとします。

例 1.

```
P セクション
     ; 6byte 命令 MOV.L #_CONST1:32, R1
     MOV.L #0FFFF8002H, R1

     ; 6byte 命令 MOV.L #_CONST2:32, R2
     MOV.L #0FFFF8006H, R2

C セクション
     _CONST0:                          ; FFFF8000h に配置
          .byte 00H,01H
     _CONST1:                          ; FFFF8002h に配置
          .byte "123"
          .byte 00H
     _CONST2:                          ; FFFF8006h に配置
          .byte "abc"
          .byte 00H
```

最適化により C セクションの配置アドレス FFFF8000H（32bit 即値）を -8000H（符号付き 16bit 即値）に出来るので、32bit 即値転送 MOV.L 命令を 16bit の即値転送命令に置き換えます。
これにより、P セクションサイズは縮小し、後続する C セクションもより小さなアドレスに移動することになります。

例 2.

```
P セクション
     ; 4byte 命令 MOV.L #_CONST1:16, R1
     MOV.L #-8002H, R1  ; 符号付き 16bit の範囲を超える。

     ; 4byte 命令 MOV.L #_CONST2:16, R2
     MOV.L #-7FFEH, R2

C セクション
     _CONST0:                        ; FFFF7FFCh に配置
         .byte 00H,01H
     _CONST1:                        ; FFFF7FFEh に配置
         .byte "123"
         .byte 00H
     _CONST2:                        ; FFFF8002h に配置
         .byte "abc"
         .byte 00H
```

結果として、複数の MOV.L の最適化の影響により、定数 CONST1 が符号付き 16bit で現せる範囲を超えた領域に配置され、次のような E0562330 エラーが発生します。

例 3.

```
     E0562330:Relocation size overflow : "fileA.obj"-"P"-"0000002"
```

(2) 問題発生箇所の推定方法

問題発生個所は次のように推定できます。

まず、最適化リンケージエディタに、nooptimize オプションと list オプションを指定してユーザプログラムをビルドし、リンク・マップ・ファイルを生成します。リンク・マップ・ファイルで "*** Mapping List ***" と書かれた行以降にセクション配置情報が出力されているので、FFFF8000h 番地以降に配置されているセクションを調べます。

例 4.

```
     *** Mapping List ***
     C
                                   ffff7ffc  ffff8005        a    1
```

次に、アセンブラに listfile オプションを指定してユーザプログラムをビルドし、ソース・リスト・ファイルを生成します。エラーメッセージが示すオブジェクトファイル（例 3 では "fileA.obj"）に対して生成されたソース・リスト・ファイルで、エラーメッセージが示すセクション（例 3 では "P"）を探します。
最適化リンケージエディタが出力するエラーメッセージ中のオフセットは最適化実施後のオフセットになります。
このため、ソース・リスト・ファイルの中では、当該オフセット（例 3 では "0000002"）以降にある命令で FFFF8000h 番地以降に配置されているセクション内のシンボルを参照する命令が対象になりますので、その命令を調べてください。
例 5 では、FFFF8000h 番地以降に配置した C セクションにある、定数 CONST1 にアクセスする命令 MOV.L が該当し、C セクションに原因があることが特定できます。

例 5.

```
                    .section P, CODE
     00000000 FB1Arrrr            MOV.L #_CONST1:16, R1 <- セクション C へのアクセス
     00000004 FB2Arrrr            MOV.L #_CONST2:16, R2

                    .section C, ROMDATA
     00000000            _CONST0:
     00000000 0001          .byte 00H,01H
     00000002            _CONST1:
     00000002 313233        .byte "123"
     00000005 00            .byte 00H
     00000006            _CONST2:
```

```
00000006 616263          .byte "abc"
00000009 00              .byte 00H
```

(3) 回避策
以下に回避策を示します。

- 回避策 1：セクションの指定を入れ替える。
最適化無しでは FFFF8000h 番地以降に配置されていて、最適化有りではセクション先頭位置が FFFF8000h 番地以前にあるセクションについて、start オプションでセクションの順番を入れ替える。

例
変更前：
```
-start=P,C,L,D/FFFF7000
```
変更後（C <-> L を入れ替え）：
```
-start=P,L,C,D/FFFF7000
```

変更後でも同様の E0562330 エラーが発生する場合があります。その場合は適宜入れ替えるセクションの組み合わせを変更してください。

- 回避策 2：FFFF8000h 番地にセクションを配置する。
最適化無しでは FFFF8000h 番地以降に配置されていて、最適化有りではセクション先頭位置が FFFF8000h 番地以前にあるセクションについて、start オプションで FFFF8000h 番地にセクションを配置する。

例
変更前：
```
-start=P,C,L,D/FFFF7000
```
変更後：
```
-start=P/FFFF7000,C,L,D/FFFF8000
```

- 回避策 3：最適化リンケージエディタの最適化を抑止する。
-nooptimize により最適化全てを抑止するか、エラーの発生有無と最適化効果（サイズ）を見ながら -optimize オプションのサブオプションを選択する。

11.5 旧バージョン・旧リビジョンとの互換性

バージョンもしくはリビジョン変更に伴う互換性に関する影響について説明します。

11.5.1 V.1.01 以降【V.1.00 との互換性】

(1) 組み込み関数の仕様変更について
アドレスを表す引数や戻り値を持つ組み込み関数については、その型を従来の unsigned long から void * に変更しました。変更になった関数を表 11.1 に示します。

表 11.1 型が変更された組み込み関数の一覧

	項目	仕様	機能	変更内容	
				箇所	内容
1	ユーザスタックポインタ (USP)	void set_usp(void *data)	USP の設定	引数	unsigned long → void *
2		void *get_usp(void)	USP の参照	戻り値	unsigned long → void *
3	割り込みスタックポインタ (ISP)	void set_isp(void *data)	ISP の設定	引数	unsigned long → void *
4		void *get_isp(void)	ISP の参照	戻り値	unsigned long → void *

	項目	仕様	機能	変更内容	
				箇所	内容
5	割り込みテーブルレジスタ(INTB)	void set_intb (void *data)	INTB の設定	引数	unsigned long → void *
6		void *get_intb(void)	INTB の参照	戻り値	unsigned long → void *
7	バックアップ PC(BPC)	void set_bpc(void *data)	BPC の設定	引数	unsigned long → void *
8		void *get_bpc(void)	BPC の参照	戻り値	unsigned long → void *
9	高速割り込みベクタレジスタ(FINTV)	void set_fintv(void *data)	FINTV の設定	引数	unsigned long → void *
10		void *get_fintv(void)	FINTV の参照	戻り値	unsigned long → void *

この変更により、V.1.00 でこれらの関数を利用されていたプログラムでは、V.1.01 では型が合わない等の警告やエラーになる場合があります。この場合は、キャストを追加または削除して型を合わせてください。
例として、V.1.00 で標準的に使用されていたスタートアッププログラム例を示します。この例は V.1.01 では W0520167 の警告メッセージが表示されますが、キャストをはずし型を合わせることで警告を回避できます。

例
set_intb 関数の利用例

```
#include <machine.h>
#pragma entry Reset_Program
void PowerON_Reset_PC(void)
{
        ...
        set_intb((unsigned long)__sectop("C$VECT")); // 警告 W0520167 になる
        ...
}
```

V.1.01 用の記述に変更した例

```
#include <machine.h>
#pragma entry Reset_Program
void PowerON_Reset_PC(void)
{
       ...
       set_intb(__sectop("C$VECT")); // キャスト (unsignd long) を削除
       ...
}
```

(2) **L** セクション追加について（**section** オプション、**Start** オプション）
V.1.01 では、文字列リテラルなどのリテラル領域を収録する L セクションを導入しました。
セクションが増えたことで、リンク時に L セクションが末尾に並ぶため、最適化リンケージエディタからアドレスエラー F0563100 が発生する場合があります。
これを回避するためには、次のいずれかの方法を採ってください。

(a) リンク時の最適化リンケージエディタの **Start** オプションに指定するセクション列に **L** を追加する

例
V.1.00 での指定例

```
-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/
0FFFF8000,C_1,C_2,C,C$*,D*,P,PIntPRG,
W*/0FFFF8100,FIXEDVECT/0FFFFFFD0
```

変更例 (C の後に L を追加する)

```
-start=B_1,R_1,B_2,R_2,B,R,SU,SI/01000,PResetPRG/
0FFFF8000,C_1,C_2,C,L,C$*,D*,P,
PIntPRG,W*/0FFFF8100,FIXEDVECT/0FFFFFFD0
```

(b) コンパイル時に **-section=L=C** を選択する
コンパイル時に -section=L=C を指定することで、リテラル領域の出力先が C セクションに変更され、V.1.00 互換のセクション構成にすることができます。
なお、上記のリンク時の Start オプションを変更する方法に比べ、コード効率に影響が出る場合があります。

11.5.2 V2.00 以降【V.1.00 ～ V.1.02 との互換性】

(1) コンパイラで -merge_files オプションを指定した場合のリンカの制限事項
コンパイラで -merge_files オプションを指定して生成したオブジェクト・モジュール・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。

(2) optimize=0 指定時の if 文に対する生成コードに関する注意事項
本バージョンでは、コンパイル時に、optimize=0 の指定の有無にかかわらず、定数だけの式からなる条件式を持つ if 文は、条件判定および実行されることのない側の文を、コード生成時に削除します。
以下に例を示します。
コメントに [削除] と書かれている行が、コード生成時に削除されます。

例 1. 定数のみの式の場合

```
int a,b,c;
void func01(void)
{
    if (1+2) {   /* [ 削除 ] */
        /* 実行される */
        a = b;
    } else {
        /* 実行されることがない */
        a++;     /* [ 削除 ] */
        b = c;   /* [ 削除 ] */
    }
}
```

例 2. シンボルアドレスを含む定数式も、定数式に含みます。

```
void f1(void),f2(void);
void func02(void)
{
    if (f1==0) {  /* [ 削除 ] */
        /* 実行されることがない */
        f2();     /* [ 削除 ] */
    } else {
        /* 実行される */
        f1();
    }
}
```

(3) -show=source の出力内容の変更
本バージョンでは、-show=source 指定時にアセンブリソースに出力される内容のうち、次の項目が変更されています。

- -debug オプションがない場合、.LINE は表示されません。
- #include の内容が展開されません。
- #line 指定に続くソース行は、命令との対応が正しくない場合があります。

11.5.3 V2.03 以降【V1.00 ～ V2.02 との互換性】

(1) const 型で初期値のない静的変数について
V2.02 以前では、初期値がある方を先に出力していましたが、本リビジョンより、静的な const 型変数は、初期値の有無にかかわらず、定義順にデータ領域に配置します。

例

```
const int a=1;
const int b;
const int c=2;
```

[V2.02.00 以前のコンパイル結果]

```
    .SECTION C,ROMDATA,ALIGN=4
_a:
    .lword  00000001H
_c:
    .lword  00000002H    ; 初期値がある方を先に出力
_b:
    .lword  00000000H
```

[V2.03.00 以降のコンパイル結果]

```
    .SECTION C,ROMDATA,ALIGN=4
_a:
    .lword  00000001H
_b:
    .lword  00000000H    ; 初期値の有無によらず、定義順に出力
_c:
    .lword  00000002H
```

11.5.4 V2.06 以降【V2.05 以前からの変更点】

(1) メモリアクセスを伴うビット操作命令の出力を制御する方法の導入
V2.05 以前のコンパイラでは、組み込み関数を除いて、ユーザがコンパイラに必ずメモリアクセスを伴うビット操作命令を出力させる方法はありませんでした。
V2.06 では、組み込み関数を使用しなくてもユーザがメモリアクセスを伴うビット操作命令を出力する / しないを制御できるように、コンパイラを改修しました。

組み込み関数を使用せずに、コンパイラにメモリアクセスを伴うビット操作命令を必ず出力させる場合は、次の条件をすべて満たすソース・プログラムを記述してください。
(a) 定数値を代入する
(b) 代入先を 1 バイト型で 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

また、コンパイラにメモリアクセスを伴うビット操作命令を出力させない場合は、上記の (c) を満たしたうえで、(a) の代入値を定数値以外にするか、または (b) の型を 1 バイト型以外の型にしてください。

上記のいずれにも該当しない場合、メモリアクセスを伴うビット操作命令を出力するかどうかは、最適化レベルやソース・プログラムの記述内容によってコンパイラが判断します。

注意 1 バイト型は、(char/unsigned char/signed char/_Bool/bool) を指します。ただし、_Bool/bool は -lang=c を指定した場合を除きます。

例

```
volatile struct {
  unsigned char bit0:1;
  unsigned int  bit1:1;
} data;

void func(void) {
  data.bit0 = 1; /* メモリアクセスを伴うビット操作命令を出力する */
  data.bit1 = 1; /* メモリアクセスを伴うビット操作命令を出力しない */
}
```

なお、V2.05 でメモリアクセスを伴うビット操作命令を必ず出力させる場合は、組み込み関数 __bclr()/__bset()/__bnot() を使用してください。組み込み関数を使用しない場合、メモリアクセスを伴うビット操作命令を出力するかどうかは、最適化レベルやソース・プログラムの記述内容によってコンパイラが判断します。
また、V2.04 以前のコンパイラでは上記の組み込み関数をサポートしていません。メモリアクセスを伴うビット操作命令が出力されたかどうかは、コンパイラが出力するアセンブリ・ソースからご確認ください。

11.5.5 コンパイラパッケージのバージョンについて

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。

11.6 W0523041 メッセージが出力される場合の注意事項 [C/C++ コンパイラ]

C 標準ヘッダをインクルードしたファイルを C++ または EC++ コンパイルしたとき、-int_to_short オプションを指定すると W0523041 メッセージが出力されることがあります。この場合は動作には問題ありませんので無視してください。

【注意】
C++ および EC++ コンパイル時は、-int_to_short オプションの指定は無効になります。
C と C++(EC++) との間で共通にアクセスするデータは、int 型ではなく long 型または short 型で宣言してください。

11.7 MVTC、POPC 命令を使用する場合の注意事項 [アセンブラ]

アセンブリ言語において、MVTC、POPC 命令に対してプログラムカウンタ (PC) は指定できません。

11.8 -delete オプションをリンク時に指定する場合の注意事項 [最適化リンケージエディタ]

-delete オプションで指定した関数シンボルが削除された場合、削除された関数定義の次の関数定義の関数名に対して、デバッグ時にエディタ上でブレークポイントを設定することができません。ラベルウィンドウからブレークポイントを設定するか、関数のプログラム行で指定してください

11.9 パス名の指定に関する注意事項

入出力ファイル指定やフォルダ指定に使用できるパス名はドライブ名を含む絶対パス、もしくは相対パスです。また、指定可能なパス名の長さは 259 文字までです。

A. クイック・ガイド

RX ファミリ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 配置領域を変更する

変数のデフォルトの配置セクションは、次のとおりになります。

- 初期値なし変数：B, B_2, B_1 セクション
- 初期値あり変数：D, D_2, D_1 セクション (ROM)、R, R_2, R_1 セクション (RAM)
- const 変数：C, C_2, C_1 セクション

配置する領域（セクション）を変更するには、#pragma section でセクション種別、セクション名を指定します。

#pragma section セクション種別　セクション名
　　変数宣言 / 定義
#pragma section
セクション種別を指定すると指定した種別のみがセクション名変更の対象になります。

尚、RX ファミリ C/C++ コンパイラでは、変数のアライメント数に応じて配置するセクションを分けています。
例)
　B　：アライメント数が 4 バイトの初期値無し変数を配置
　B_2：アライメント数が 2 バイトの初期値無し変数を配置
　B_1：アライメント数が 1 バイトの初期値無し変数を配置

初期値あり変数の場合は、初期値が ROM に配置されて、変数自体は RAM に配置されます（ROM/RAM 両方の領域が必要になります）。スタートアップ・ルーチンの resetprg.c ファイルを使用した場合は、INITSCT 関数で ROM の初期値を RAM の変数にコピーします。

セクション種別と生成されるセクションの関係は次のとおりです。

表 A.1　　セクション一覧

名称	セクション名称	属性	形式種別	初期値 / 書き込み	アライメント
定数領域	C[*1*2]	romdata	相対	有 / 不可	4byte
	C_2[*1*2]	romdata	相対	有 / 不可	2byte
	C_1[*1*2]	romdata	相対	有 / 不可	1byte
初期化データ	D[*1*2]	romdata	相対	有 / 可	4byte
	D_2[*1*2]	romdata	相対	有 / 可	2byte
	D_1[*1*2]	romdata	相対	有 / 可	1byte
未初期化データ	B[*1*2]	data	相対	無 / 可	4byte
	B_2[*1*2]	data	相対	無 / 可	2byte
	B_1[*1*2]	data	相対	無 / 可	1byte
switch 文分岐テーブル領域	W[*1]	romdata	相対	有 / 不可	4byte
	W_2[*1]	romdata	相対	有 / 不可	2byte
	W_1[*1]	romdata	相対	有 / 不可	1byte
C++ 初期処理後処理データ	C$INT	romdata	相対	有 / 不可	4byte

名称	セクション名称	属性	形式種別	初期値 / 書き込み	アライメント
C++ 仮想関数表	C$VTBL	romdata	相対	有 / 不可	4byte
絶対アドレス変数	$ADDR_ <section>_ <address> *3	data	絶対	有無 / 可不可 *4	-
可変ベクタ領域	C$VECT	romdata	相対	無 / 可	-

注 1. section オプションまたは拡張子 #pragma section でセクション名を切り替えることができます。
ただし、文字列リテラルなどデータの一部に #pragma section の影響を受けないものがあります。
詳しくは、4.2.3 #pragma 指令の #pragma section の詳細説明を参照ください。

注 2. セクション名切り替えの際に、アライメント数が 4 のセクションを指定することで、アライメントが 1 または 2 のセクション名も変更されます。#pragma endian で endian オプションと異なる指定のエンディアンを指定した場合、#pragma endian big であれば _B を、#pragma endian little であれば _L を、セクション名の後ろに付加した専用のセクションを生成し、該当データを格納します。
ただし、文字列リテラルなどデータの一部に #pragma endian の影響を受けないものがあります。
詳しくは、4.2.3 #pragma 指令の #pragma endian の詳細説明を参照ください。

注 3. <section> は C, D, B のセクション名称、<address> は絶対アドレス値 (16 進数) になります。

注 4. 初期値、書き込み操作は <section> の属性に従います。

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からはずされ、レジスタに割り付ける最適化などを行わなくなります。volatile 指定された変数に対する操作を行うときは、必ずメモリから値を読み込み、操作後にメモリへ値を書き込むコードになります。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。

A.1.3 変数を宣言したサイズでアクセスするコードを生成する

変数を宣言したサイズでアクセスする場合は、__evenaccess の拡張機能を使用します。
__evenaccess の宣言により変数の型のサイズでアクセスすることを保証します。保証対象サイズは、4 バイト以下の整数スカラ型 (signed char, unsigned char, signed short, unsigned short, signed int, unsigned int, signed long, unsigned long) です。
構造体や共用体単位でアクセスする場合は、__evenaccess の指定は無効です。
構造体または共用体に指定した場合、全てのメンバに __evenaccess を指定したのと同じ効果になります。その場合、4 バイト以下の整数スカラ型メンバのアクセスサイズは保証しますが、構造体または共用体単位でのアクセスサイズは保証しません。
【記述例】
C ソースコード

```
#pragma address A=0xff0178
unsigned long __evenaccess A;
void test(void)
{
      A &= ~0x20;
}
```

出力コード（__evenaccess 非指定時）

```
_test:
        MOV.L #16712056,R1
        BCLR #5,[R1] ; 1 バイトメモリアクセス
        RTS
```

出力コード（__evenaccess 指定時）

```
_test:
        MOV.L #16712056,R1
        MOV.L [R1],R5 ; 4 バイトメモリアクセス
        BCLR #5,R5
        MOV.L R5,[ R1] ; 4 バイトメモリアクセス
        RTS
```

A.1.4 値を変更しない初期化変数は const 宣言をする

初期値のある変数は、通常、起動時に ROM エリアから RAM エリアに転送して、RAM エリアを使って処理を行います。このため、プログラム内で値が不変な初期化データの場合、確保した RAM エリアが無駄になります。初期化データに const 演算子をつけておくと、起動時の RAM エリアへの転送が抑止され、使用メモリ量の節約になります。
また初期値は変更しない、というルールでプログラムを作成すると、ROM イメージの作成 が容易になります。

【改善前記述例】
char a[] = { 1, 2, 3, 4, 5 };
初期値を ROM から RAM へ転送して処理を行います。

【改善後記述例】
const char a[] = { 1, 2, 3, 4, 5 };
ROM 上の初期値を使用して処理を行います。

A.1.5 const 定数ポインタを定義する

ポインタについては、“const” の指定場所により、異なる解釈がされます。
【記述例 1】
  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（B セクション）に配置されます。
  p = 0 ; / エラー */
  p = 0 ; /* 正しい */
【記述例 2】
  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（C セクション）に配置されます。
  p = 0 ; / 正しい */
  p = 0 ; /* エラー */
【記述例 3】
  const char *const p ;
ポインタ自体（p）、ポインタが示すオブジェクト（*p）を書き換えできないことを示します。
したがって、以下のようになり、ポインタ自体は ROM（C セクション）に配置されます。
  p = 0 ; / エラー */
  p = 0 ; /* エラー */

A.1.6 セクションのアドレスを参照する

セクションアドレス演算子を使用することでセクションのアドレスやサイズを参照することができます。
__sectop ("< セクション名 >") < セクション名 > の先頭アドレスを参照します。
__secend ("< セクション名 >") < セクション名 > の先頭に < セクション名 > のサイズを加算した値を参照します。
__secsize ("< セクション名 >") < セクション名 > のサイズを生成します。

【記述例】

```
#pragma section $DSEC
static const struct {
        void *rom_s; /* 初期化データセクションの ROM 上の先頭アドレス値を取得 */
        void *rom_e; /* 初期化データセクションの ROM 上の最終アドレス値を取得 */
        void *ram_s; /* 初期化データセクションの RAM 上の先頭アドレス値を取得 */
} DTBL[]={__sectop("D"), __secend("D"), __sectop("R")};
```

スタートアップ・ルーチンの resetprg.c ファイルの INITSCT 関数内では、ROM から RAM への転送、及び未初期化領域の初期化を実行します。この実行の際、dbsct.c ファイル内に記述されている __sectop、__secend で取得したアドレスを参照しています。

A.2 関数

この節では、関数について説明します。

A.2.1 アセンブラ命令の埋め込み

RX C/C++ コンパイラでは、#pragma inline_asm において、C 言語ソース・プログラム中にアセンブラ命令が記述できます。

【記述例】

```
#pragma inline_asm func
static int func(int a, int b){
         ADD R2,R1 ; アセンブリ記述
}
main(int *p){
         *p = func(10,20);
}
```

#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。

A.2.2 関数のインライン展開を行う

#pragma inline は、インライン展開する関数を宣言します。
インライン展開の有無はコンパイラオプションの inline/noinline でも制御しますが、noinline オプションが指定された場合でも、#pragma inline 指定された関数はインライン展開の対象となります。
関数名には、グローバル関数および静的関数メンバを指定できます。#pragma inline で指定した関数名の関数と関数指定子 inline(C++ 言語および C(C99) 言語) を指定した関数がインライン展開されると、その関数を呼び出したところに関数の本体が展開されます。
【記述例】
C ソースコード

```
#pragma inline(func)
static int func (int a, int b)
{
          return (a+b)/2;
}
int x;
main()
{
          x=func(10,20);
}
```

展開イメージ

```
int x;
main()
{
         int func_result;
         {
                    int a_1=10, b_1=20;
                    func_result=(a_1+b_1)/2;
         }
           x=func_result;
}
```

A.2.3 関数のインライン展開を行う (ファイル間)

通常、インライン展開はファイル内の関数のみがインライン展開の対象になりますが、コンパイラの -inline および -merge_files オプションを使用することでファイル間の関数呼び出しもインライン展開の対象になります。
【記述例】

```
<a.c>
  func(){
      g();
  }
<b.c>
  g(){
      h();
  }
ccrx -merge_files -inline a.c b.c
```

と指定してコンパイルすることにより a.c の中の関数 g の呼び出しが展開され以下のようになります。

```
func(){
    h();
}
```

A.3 マイコン機能の使用

この節では、マイコン機能の使用について説明します。

A.3.1 C 言語で割り込み処理を行う

割り込み関数は、#pragma interrupt によって宣言します。
【記述例】
C ソース

```
#pragma interrupt func
void func(){ .... }
```

生成コード

```
_func:
          PUSHM R1-R3 ; 関数内で使用しているレジスタを退避
                  ・・・
          (R1,R2,R3 を関数内で使用 )
                  ・・・
          POPM R1-R3 ; 入口で退避したレジスタを回復
          RTE
```

A.3.2 C 言語で CPU 命令を使用する

制御レジスタへアクセスする場合や C 言語で表現できない特殊命令に関しては、組込み関数を提供しています。「4.2.6 組み込み関数」を参照してください。

A.4 変数（アセンブラ）

この節では、変数（アセンブラ）について説明します。

A.4.1 初期値なし変数を定義する

DATA セクション内にメモリ領域を確保します。
DATA セクションを定義するには、.SECTION 指示命令を使用し、メモリ領域には、1 バイト単位の場合には .BLKB 指示命令を、2 バイト単位の場合には .BLKW 指示命令を、4 バイト単位の場合には .BLKL 指示命令を、8 バイト単位の場合には .BLKD 指示命令を使用します。
【記述例】

```
        .SECTION area,DATA
work1:  .BLKB 1; 1 バイトの単位で RAM 領域を確保
work2:  .BLKW 1; 2 バイトの単位で RAM 領域を確保
work3:  .BLKL 1; 4 バイトの単位で RAM 領域を確保
work4:  .BLKD 1; 8 バイトの単位で RAM 領域を確保
```

A.4.2 初期値あり const 定数を定義する

ROMDATA セクション内のメモリ領域を初期化します。
ROMDATA セクションを定義するには .SECTION 指示命令を使用し、メモリ初期化には、1 バイトの場合には .BYTE 指示命令を、2 バイトの場合には .WORD 指示命令を、4 バイトの場合には .LWORD 指示命令を、浮動小数点の 4 バイトの場合には .FLOAT 指示命令を、浮動小数点の 8 バイトの場合には .DOUBLE 指示命令を使用します。
【記述例】

```
       .SECTION value,ROMDATA
work1: .BYTE "data"  ;  1 バイト長の固定データを ROM に格納
work2: .WORD "data"  ;  2 バイト長の固定データを ROM に格納
work3: .LWORD "data" ;  4 バイト長の固定データを ROM に格納
work4: .FLOAT 5E2    ;  4 バイト長の浮動小数点データを ROM に格納
work5: .DOUBLE 5E2   ;  8 バイト長の浮動小数点データを ROM に格納
```

A.4.3 セクションのアドレスを参照する

SIZEOF、TOPOF 演算子によりオペランドに指定したセクションのサイズ、開始アドレスを値として扱います。
【記述例】

```
・・・
MVTC        #(TOPOF SU + SIZEOF SU),USP
; SU の開始アドレス +SU のサイズによりユーザスタック領域のアドレスを USP へ設定
MVTC        #(TOPOF SI + SIZEOF SI),ISP
; SI の開始アドレス +SI のサイズにより割り込みスタック領域のアドレスを ISP へ設定
・・・
```

A.5 スタートアップ・ルーチン

この節では、スタートアップ・ルーチンについて説明します。

A.5.1 スタック領域を確保する

スタートアップ・ルーチンの resetprg.c ファイルの PowerON_Reset_PC 関数を "#pragma entry" 宣言している事により、下記の設定によりコンパイラとリンカが自動的にユーザスタック USP/ 割り込みスタック ISP 初期化コードを関数先頭に生成します。

(1) ユーザスタックの設定
stacksct.h ファイルの #pragma stacksize su=0xXXX でスタック領域のサイズ、最適化リンカの -start オプションで SU セクションの配置を指定します。

(2) 割り込みスタックの設定
stacksct.h ファイルの #pragma stacksize si=0xXXX でスタック領域のサイズ、最適化リンカの -start オプションで SI セクションの配置を指定します。

【記述例】

```
<resetprg.c>
 ・・・
 #pragma section ResetPRG
 #pragma entry PowerON_Reset_PC
 void PowerON_Reset_PC(void)
 {
 ・・・
<stacksct.h>
 #pragma stacksize su=0x300
 #pragma stacksize si=0x100
```

【生成コード例】

```
 // -start=SU,SI/01000 と指定した場合
_PowerON_Reset_PC          MVTC        #00001300H,USP
                           MVTC        #00001400H,ISP
                            ・・・
```

A.5.2 RAM を初期化する

スタートアップ・ルーチンの resetprg.c ファイルの _INITSCT 関数内で未初期化領域の初期化を行います。初期化の対象となるセクションの追加は、dbsct.c ファイル内の下記に記述を追記することで可能です。

【記述例】

```
<dbsct.c>
  ・・・
  #pragma section C C$BSEC
  extern const struct {
     _UBYTE *b_s;   /* Start address of non-initialized data section */
     _UBYTE *b_e;   /* End address of non-initialized data section */
  } _BTBL[] = {
       { __sectop("B"), __secend("B") },
       { __sectop("B_2"), __secend("B_2") },
       { __sectop("B_1"), __secend("B_1") }
  };
  ・・・
```

上記では、B, B_2, B_1 セクションを初期化するために INITSCT で使用するアドレスをテーブルに格納します。

A.5.3 初期値あり変数を ROM から RAM へ転送する

スタートアップ・ルーチンの resetprg.c ファイルの _INITSCT 関数内で初期値あり変数を ROM から RAM への転送を行います。転送の対象となるセクションの追加は、dbsct. c ファイル内の下記に記述を追加することで可能です。

【記述例】

```
<dbsct.c>
   ・・・
   #pragma section C C$DSEC
   extern const struct {
      _UBYTE *rom_s;      /* Start address of the initialized data section in ROM */
      _UBYTE *rom_e;      /* End address of the initialized data section in ROM */
      _UBYTE *ram_s;      /* Start address of the initialized data section in RAM */
   }   _DTBL[] = {
          { __sectop("D"), __secend("D"), __sectop("R") },
          { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
          { __sectop("D_1"), __secend("D_1"), __sectop("R_1") }
   };
   ・・・
```

上記では、D, D_2, D_1 セクションを R, R_2, R_1 セクションへの転送するために INITSCT 関数で使用するアドレスをテーブルに格納します。尚、D, D_2, D_1 と R, R_2, R_1 の配置アドレスは、最適化リンカの -start オプションで指定し、ROM から RAM への転送によるリロケーション解決は、最適化リンカの -rom オプションで指定します。

A.6 コードサイズの削減

この節では、コードサイズの削減について説明します。

A.6.1 データの構造

関連するデータを同一関数の中で何度も参照している場合、構造体を用いると相対アクセスを利用したコードが生成され易くなり、効率向上が期待できます。また、引数として渡す場合も効率が向上します。相対アクセスにはアクセス範囲に制限があるため、頻繁にアクセスするデータは構造体の先頭に集めると効果的です。

データを構造化すると、データの表現を変更するようなチューニングが容易になります。

【使用例】

変数 a, b, c に数値を代入します。

改善前ソースコード

```
  int a, b, c;
  void func()
  {
         a = 1;
         b = 2;
         c = 3;
 }
```

改善前アセンブリ展開コード

```
 _ func:
         MOV.L #_a,R4
         MOV.L #00000001H,[R4]
         MOV.L #_b,R4
         MOV.L #00000002H,[R4]
         MOV.L #_c,R4
         MOV.L #00000003H,[R4]
         RTS
```

改善後ソースコード

```
struct s{
        int a;
        int b;
        int c;
} s1;
void func()
{
        register struct s *p=&s1;
        p->a = 1;
        p->b = 2;
        p->c = 3;
}
```

改善後アセンブリ展開コード

```
_func:
       MOV.L #_s1,R5
       MOV.L #00000001H,[R5]
       MOV.L #00000002H,04H[R5]
       MOV.L #00000003H,08H[R5]
       RTS
```

A.6.2 局所変数と大域変数

局所変数として使用できるものは、大域変数として宣言しないで必ず局所変数として宣言してください。大域変数は、関数呼び出しやポインタ操作によって値が変化してしまう可能性があるため、最適化の効率が悪くなります。

局所変数を使用すると次の利点があります。

a. アクセス効率が良い。

b. レジスタに割り付けられる可能性がある。

c. 最適化の効率が良い

【使用例】

一時変数に大域変数を使った場合 (改善前) と局所変数を使った場合 (改善後)

改善前ソースコード

```
int tmp;
void func(int* a, int* b)
{
         tmp = *a;
         *a = *b;
         *b = tmp;
}
```

改善前アセンブリ展開コード

```
__func:
        MOV.L #_tmp,R4
        MOV.L [R1],[R4]
        MOV.L [R2],[R1]
        MOV.L [R4],[R2]
        RTS
```

改善後ソースコード

```
void func(int* a, int* b)
{
       int tmp;
       tmp = *a;
       *a = *b;
       *b = tmp;
}
```

改善後アセンブリ展開コード

```
__func:
        MOV.L [R1],R5
        MOV.L [R2],[R1]
        MOV.L R5,[R2]
        RTS
```

A.6.3 構造体宣言のメンバオフセット

構造体メンバは、構造体アドレスにオフセットを加算してアクセスします。オフセットを小さくするとサイズが有利になるので、よく使用するメンバを先頭に宣言するようにしてください。

もっとも効果的なのは、signed char, unsigned char 型で先頭から 32byte 未満 , short, unsigned short 型で先頭から 64byte 未満 , int, unsigned, long, unsigned long 型で先頭から 128byte 未満です。

【使用例】

以下の例は、構造体のオフセットによってコードが変わる例を示します。

改善前ソースコード

```
struct str {
         long L1[8];
         char C1;
};
struct str STR1;
char x;
void func()
{
         x = STR1.C1;
}
```

改善前アセンブリ展開コード

```
_func:
       MOV.L #_STR1,R4
       MOVU.B 20H[R4],R5
       MOV.L #_x,R4
       MOV.B R5,[R4]
       RTS
```

改善後ソースコード

```
struct str {
         char C1;
         long L1[8];
};
struct str STR1;
char x;
void func()
{
         x = STR1.C1;
}
```

改善後アセンブリ展開コード

```
__func:
        MOV.L #_STR1,R4
        MOVU.B [R4],R5
        MOV.L #_x,R4
        MOV.B R5,[R4]
        RTS
```

■注意事項
構造体を定義する際には、境界調整数を意識してメンバの宣言をおこなってください。
構造体の境界調整数は構造体内の最も大きな境界調整値に合わせられ、構造体のサイズは境界調整数の倍数となります。その為、構造体の末尾が構造体自身の境界調整数と合わない場合に、次の境界調整を保証するために生成される、未使用領域もサイズに含めてしまいます。

改善前ソースコード

```
/* 最大メンバが int 型の為、境界調整数は 4 */
struct str {
          char C1; /* 1byte + 境界調整分 3byte */
          long L1; /* 4byte */
          char C2; /* 1byte */
          char C3; /* 1byte */
          char C4; /* 1byte + 境界調整分 1byte */
}STR1;
```

改善前 str サイズ

```
        .SECTION B,DATA,ALIGN=4
        .glb _STR1
_STR1:                      ; static: STR1
        .blkl 3
```

改善後ソースコード

```
/* 最大メンバが int 型の為、境界調整数は 4 */
struct str {
          char C1; /* 1byte */
          char C2; /* 1byte */
          char C3; /* 1byte */
          char C4; /* 1byte */
          long L1; /* 4byte */
}STR1;
```

改善後 str サイズ

```
        .SECTION B,DATA,ALIGN=4
        .glb _STR1
_STR1:                      ; static: STR1
        .blkl 2
```

A.6.4 ビットフィールドの割り付け

異なるビットフィールドのメンバを設定するためには、そのたびにビットフィールドを含むデータにアクセスしなければなりません。関連するビットフィールドを同じ構造体内にまとめて割り付けることによって、このアクセスを一度で済ますことができます。

【使用例】
同じ構造体に関連するビットフィールドを割り付けることによってサイズが改善する例を示します。

改善前ソースコード

```
struct str
{
          Int flag1:1;
}b1,b2,b3;
void func()
{
          b1.flag1 = 1;
          b2.flag1 = 1;
          b3.flag1 = 1;
}
```

改善前アセンブリ展開コード

```
_func:
        MOV.L #_b1,R5
        BSET #00H,[R5]
        MOV.L #_b2,R5
        BSET #00H,[R5]
        MOV.L #_b3,R5
        BSET #00H,[R5]
        RTS
```

改善後ソースコード

```
struct str
{
            int flag1:1;
            int flag2:1;
            int flag3:1;
}a1;
void func()
{
            a1.flag1 = 1;
            a1.flag2 = 1;
            a1.flag3 = 1;
}
```

改善後アセンブリ展開コード

```
__func:
         MOV.L #_a1,R4
         MOVU.B [R4],R5
         OR #07H,R5
         MOV.B R5,[R4]
         RTS
```

A.6.5 ベースレジスタ指定時の外部変数アクセス最適化

RAM セクションのベースレジスタに R13 を指定した場合、RAM セクションへのアクセスが、R13 レジスタ相対となります。更にモジュール間の外部変数アクセス最適化を有効にした場合、R13 レジスタ相対値が最適化されて、8bit 範囲以下の値であれば、命令サイズが小さくなる場合があります。

【使用例】

改善前ソースコード

```
int a;
int b;
int c;
int d;
void fu{
        a=0;
        b=1;
        c=2;
        d=3;
}
```

改善前アセンブリ展開コード

```
_func:
        MOV.L #_a,R4
        MOV.L #0000000H,[R4]
        MOV.L #_b,R4
        MOV.L #00000001H,{R4}
        MOV.L #_c,R4
        MOV.L #00000002H,[R4]
        MOV.L #_d,[R4]
        MOV.L #00000003H,[R4]
        RTS
```

改善後ソースコード

```
int a;
int b;
int c;
int d;
void fu{
        a=0;
        b=1;
        c=2;
        d=3;
}
```

改善後アセンブリ展開コード

```
__func:
         MOV.L #0000000H,_a-__RAM_TOP:16[R13]
         MOV.L #0000001H,_b-__RAM_TOP:16[R13]
         MOV.L #0000002H,_c-__RAM_TOP:16[R13]
         MOV.L #0000003H,_d-__RAM_TOP:16[R13]
         RTS
```

A.6.6 外部変数アクセス最適化時のリンカのセクションアドレス指定順

レジスタ相対形式でメモリにアクセスする命令では、ディスプレースメント値が小さいほうが、命令サイズが小さくなります。
以下の指標を参考にリンカでのセクション割り付け順を変更するとコードサイズを改善できる場合があります。

- 関数内でのアクセス回数の多い外部変数のセクションを前にする。

- 型サイズの小さい外部変数のセクションを前にする。

但し、外部変数アクセス最適化は、コンパイラが 2 度実行されるのでビルド時間は長くなります。
【使用例】
改善前ソースコード

```
/* D_1 セクション */
char d11=0, d12=0, d13=0, d14=0;
/* D_2 セクション */
short d21=0, d22=0, d23=0, d24=0, dmy2[12]={0};
/* D セクション */
int d41=0, d42=0, d43=0, d44=0, dmy4[60]={0}

void func(int a){
        d11 = a;
        d12 = a;
        d13 = a;
        d14 = a;
        d21 = a;
        d22 = a;
        d23 = a;
        d24 = a;
        d41 = a;
        d42 = a;
        d43 = a;
        d44 = a;
}
```

改善前アセンブリ展開コード

```
< セクションの割り付け順を D, D_2, D_1  または D* とした場合 >
_func:
        MOV.L #d41,R4
        MOV.B R1,0120H[R4]
        MOV.B R1,0121H[R4]
        MOV.B R1,0122H[R4]
        MOV.B R1,0123H[R4]
        MOV.W R1,0100H[R4]
        MOV.W R1,0102H[R4]
        MOV.W R1,0104H[R4]
        MOV.W R1,0106H[R4]
        MOV.L R1,[R4]
        MOV.L R1,04H[R4]
        MOV.L R1,08H[R4]
        MOV.L R1,0CH[R4]
        RTS
```

改善後ソースコード

```
/* D_1 セクション */
char d11=0, d12=0, d13=0, d14=0;
/* D_2 セクション */
short d21=0, d22=0, d23=0, d24=0, dmy2[12]={0};
/* D セクション */
int d41=0, d42=0, d43=0, d44=0, dmy4[60]={0}

void func(int a){
        d11 = a;
        d12 = a;
        d13 = a;
        d14 = a;
        d21 = a;
        d22 = a;
        d23 = a;
        d24 = a;
        d41 = a;
        d42 = a;
        d43 = a;
        d44 = a;
}
```

改善後アセンブリ展開コード

```
< セクションの割り付け順を D_1, D_2, D  または D* とした場合 >
_func:
       MOV.L #d11,R4
       MOV.B R1,[R4]
       MOV.B R1,01H[R4]
       MOV.B R1,02H[R4]
       MOV.B R1,03H[R4]
       MOV.W R1,04H[R4]
       MOV.W R1,06H[R4]
       MOV.W R1,08H[R4]
       MOV.W R1,0AH[R4]
       MOV.L R1,24H[R4]
       MOV.L R1,28H[R4]
       MOV.L R1,2CH[R4]
       MOV.L R1,30H[R4]
       RTS
```

A.6.7 割り込み

割り込み処理の前後では多くのレジスタ退避・回復が発生し、期待する割り込み応答時間を得られない場合があります。その場合は、高速割り込み指定（fint）と fint_register オプションを利用することで、レジスタの退避・回復を抑えることができ、割り込み応答時間の短縮を図ることができます。

ただし、fint_register オプションを利用すると他の関数での使用可能なレジスタが減るため、プログラム全体での効率が低下する場合があるのでご注意ください。

【使用例】

改善前ソースコード

```
#pragma interrupt int_func
volatile int count;

void  int_func()
{
        count++;
}
```

改善前アセンブリ展開コード

```
_int_func:
          PUSHM R4-R5
          MOV.L #_count,R4
          MOV.L [R4],R5
          ADD #01H,R5
          MOV.L R5,[R4]
          POPM R4-R5
          RTE
```

改善後ソースコード

```
#pragma interrupt int_func(fint)
volatile int count;

void  int_func()
{
        count++;
}
```

改善後アセンブリ展開コード

```
<fint_register=2 オプション指定時 >
_int_func:
           MOV.L #_count,R12
           MOV.L [R12],[R13]
           ADD #01H,R13
           MOV.L R13,[R12]
           RTFI
```

A.7 処理の高速化

この節では、処理の高速化について説明します。

A.7.1 ループ制御変数

ループ終了条件の判定で、サイズの違いによりループ制御変数がその比較対象のデータを表現できない可能性のある場合は、ループ展開最適化がかかりません。たとえばループ制御変数が signed char で比較対象のデータが signed long の場合はループ展開最適化がかかりません。

そのため、signed char, signed short に比べ、signed long の方がループ展開最適化を適用し易くなります。ループ展開最適化を活用したい場合はループ制御変数を 4 バイト整数型としてください。

【使用例】

改善前ソースコード

```
signed long array_size=16;
signed char array[16];

void func()
{
        signed char i;
        for(i=0;i<array_size;i++)
        {
                    array[i]=0;
        }
}
```

改善前アセンブリ展開コード

```
<loop=2 指定時 >
_func:
        MOV.L #_array_size,R4
        MOV.L [R4],R2
        MOV.L #00000000H,R5
        BRA L11
L12:
        MOV.L #_array,R14
        MOV.L #00000000H,R3
        MOV.B R3,[R5,R4]
        ADD #01H,R5
L11:
        MOV.B R5,R5
        CMP R2,R5
        BLT L12
L13:
        RTS
```

改善後ソースコード

```
signed long array_size=16;
signed char array[16];

void func()
{
         signed long i;
         for(i=0;i<array_size;i++)
         {
                     array[i]=0;
         }
}
```

改善後アセンブリ展開コード

```
<loop=2 指定時 >
_func:
        MOV.L #_array_size,R 5
        MOV.L [R5],R2
        MOV.L #00000000H,R 4
        ADD #0FFFFFFFFH,R2,R3
        CMP R3,R2
        BLE L12
L11:
        MOV.L #_array,R1
        MOV.L R1,R5
        BRA L13
L14:
        MOV.W #0000H,[R5]
        ADD #02H,R5
        ADD #02H,R4
L13:
        CMP R3,R4
        BLT L14
L15:
        CMP R2,R4
        BGE L17 L16:
        MOV.L #00000000H,R5
        MOV.B R5,[R4,R1]
        RTS
L12:
        MOV.L #_array,R5
        MOV.L #00000000H,R3
L19:
        CMPR2,R4
        BGE L17
L20:
        MOV.B R3,[R5+]
        ADD #01H,R4
        BRA L19
L17:
        RTS
```

A.7.2 関数のインタフェース

引数がすべてレジスタに乗るように (4 個まで) 引数の数を厳選してください。引数が多い場合は、構造体にしてポインタで渡してください。もし、構造体のポインタではなく、構造体そのものを受け渡すとレジスタに乗らない場合があります。引数がレジスタに乗れば、呼び出し、関数の出入り口の処理が簡単になります。またスタック領域も節約できます。なお、レジスタは R1 ～ R4 が引数用です。

【使用方法】

関数 f の引数が引数用レジスタ個数よりも 4 個多くあります。

改善前ソースコード

```
void call_func()
{
          func(1,2,3,4,5,6,7,8);
}
```

改善前アセンブリ展開コード

```
_call_func:
            SUB #04H,R0
            MOV.L #08070605H,[R0]
            MOV.L #00000004H,R4
            MOV.L #00000003H,R3
            MOV.L #00000002H,R2
            MOV.L #00000001H,R1
            BSR _func
            ADD #04H,R0
            RTS
```

改善後ソースコード

```
struct str{
         char a;
         char b;
         char c;
         char d;
         char e;
         char f;
         char g;
         char h;
};
struct str arg = {1,2,3,4,5,6,7,8};

void call_func()
{
          func(&arg);
}
```

改善後アセンブリ展開コード

```
_call_func:
         MOV.L #arg,R1
         BRA _func
```

A.7.3 ループ回数の削減

ループの展開は特に内側のループが有効です。ループの展開によりプログラムサイズは増大するので、プログラムサイズを犠牲にしても実行速度を向上させたい場合に適用してください。

【使用例】

配列 a[] を初期化します。

改善前ソースコード

```
extern int a[100];
void func()
{
        int I;
        for( i = 0 ; i < 100 ; i++ ){
                    a[i] = 0;
        }
}
```

改善前アセンブリ展開コード

```
_func:
        MOV.L #00000064H,R4
        MOV.L #_a,R5
        MOV.L #00000000H,R3
L11:
        MOV.L R3,[R5+]
        SUB #01H,R4
        BNE L11
L12:
        RTS
```

改善後ソースコード

```
extern int a[100];
void func()
{
         int I;
         for( i = 0 ; i < 100 ; i+=2 )
         {
              a[i] = 0;
              a[i+1] = 0;
         }
}
```

改善後アセンブリ展開コード

```
_func:
        MOV.L #00000032H,R4
        MOV.L #_a,R5
L11:
        MOV.L #00000000H,[R5]
        MOV.L #00000000H,04H[R5]
        ADD #08H,R5
        SUB #01H,R4
        BNE L11
L12:
        RTS
```

A.7.4 テーブルの活用

switch 文の各 case の処理がほぼ同じ場合は、テーブルを使用できないか検討してください。
【使用方法】
変数 i の値により変数 ch に代入する文字定数を変えます。
改善前ソースコード

```
char func(int i)
{
         char ch;
         switch (i) {
                   case 0:
                            ch = ‘a’; break;
                   case 1:
                            ch = ‘x’; break;
                   case 2:
                            ch = ‘b’; break;
         }
         return(ch);
}
```

改善前アセンブリ展開コード

```
_func:
          CMP #00H,R1
          BEQ L17
L16:
          CMP #01H,R1
          BEQ L19
          CMP #02H,R1
          BEQ L20
          BRA L21
L12:
L17:
          MOV.L #00000061H,R1
          BRA L21
L13:
L19:
          MOV.L #00000078H,R1
          BRA L21
L14:
L20:
          MOV.L #00000062H,R1
L11:
L21:
          MOVU.B R1,R1
          RTS
```

改善後ソースコード

```
char chbuf[] = { ‘a’ , ‘x’ , ‘b’ };

char func(int i)
{
          return (chbuf[i]);
}
```

改善後アセンブリ展開コード

```
_f
           MOV.L #_chbuf,R4
           MOVU.B [R1,R4],R1
           RTS
```

A.7.5 分岐

else if 文のように上から順に比較をする場合、場合分けが増えると末端のケースの実行速度は低下します。頻繁に分岐するケースは先頭近くに配置してください。

【使用例】

引数の値によりリターン値を変えます。

改善前ソースコード

```
int func(int a)
{
        if (a==1)
                a = 2;
        else if (a==2)
                a = 4;
        else if (a==3)
                a = 0;
        else
                a = 0;
        return(a);
}
```

改善前アセンブリ展開コード

```
_func:
        CMP #01H,R1
        BEQ L11
L12:
        CMP #02H,R1
        BNE L14
L13:
        MOV.L #00000004H,R1
        RTS
L14:
        CMP #03,R1
        BNE L17
L16:
        MOV.L #00000008H,R1
        RTS
L17:
        MOV.L #00000000H,R1
        RTS
L11:
        MOV.L #00000002H,R1
        RTS
```

改善後ソースコード

```
int func(int a)
{
        if (a==3)
                a = 8;
        else if (a==2)
                a = 4;
        else if (a==1)
                a = 2;
        else
                a = 0;
        return (a);
}
```

改善後アセンブリ展開コード

```
_func:
        CMP #03H,R1
        BEQ L11
L12:
        CMP #02H,R1
        BNE L14
L13:
        MOV.L #00000004H,R1
        RTS
L14:
        CMP #01H,R1
        BNE L17
L16:
        MOV.L #00000002H,R1
        RTS
L17:
        MOV.L #00000000H,R1
        RTS
L11:
        MOV.L #00000008H,R1
        RTS
```

A.7.6 インライン展開

頻繁によびだされる関数をインライン展開することにより、実行速度の向上が図れます。特にループ内で呼ばれる関数などを展開すると大きな効果を得られる場合もあります。しかし、インライン展開をした場合、プログラムサイズが増大する傾向にありますので、プログラムサイズを犠牲にしても実行速度を向上させたい場合に適用してください。

【使用例】

配列 a と配列 b の要素を交換します。

改善前ソースコード

```
int x[10], y[10];
static void sub(int *a, int *b, int i)
{
        int temp;
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
}

void func()
{
        int i;
        for(i=0;i<10;i++)
        {
                sub(x,y,i);
        }
}
```

改善前アセンブリ展開コード

```
__$sub:
        SHLL #02H,R3
        ADD R3,R1
        MOV.L [R1],R5
        ADD R3,R2
        MOV.L [R2],[R1]
        MOV.L R5,[R2]
        RTS
_func:
        PUSHM R6-R8
        MOV.L #00000000H,R6
        MOV.L #_x,R7
        MOV.L #_y,R8
L12:
        MOV.L R6,R3
        MOV.L R7,R1
        MOV.L R8,R2
        ADD #01H,R6
        BSR __$sub
        CMP #0AH,R6
        BLT L12
L13:
        RTSD #0CH,R6-R8
```

改善後ソースコード

```
int x[10], y[10];
#pragma inline(sub)
static void sub(int *a, int *b, int I)
{
         int temp;
         temp = a[i];
         a[i] = b[i];
         b[i] = temp;
}

void func()
{
         int I;
         for(i=0;i<10;i++)
         {
                   sub(x,y,i);
         }
}
```

改善後アセンブリ展開コード

```
;  インライン展開により
;  _sub のコードが削減されている
_func:
        MOV.L #0000000AH,R1
        MOV.L #_y,R2
        MOV.L #_x,R3
L11:
        MOV.L [R3],R4
        MOV.L [R2],R5
        MOV.L R4,[R2+]
        MOV.L R5,[R3+]
        SUB #01H,R1
        BNE L11
L12:
        RTS
```

A.8 C ソースの修正

拡張機能を使用することにより、効率の良いオブジェクトを生成することができます。
ここでは、他のコンパイラから CC-RX への移植と、CC-RX から他の C コンパイラへの移植の 2 つの場合について、その方法を説明します。

＜他のコンパイラから CC-RX ＞

- #pragma
  他のコンパイラが #pragma をサポートしている場合は、ソースを修正する必要があります。修正方法は、そのコンパイラの仕様によって検討します。
- 拡張仕様
  他のコンパイラがキーワードを追加するなどの仕様の拡張を行っている場合は、修正する必要があります。修正方法はそのコンパイラの仕様によって検討します。

注　ANSI でサポートされている前処理指令の 1 つで、#pragma に続く文字列をコンパイラへの指令として認識させるものです。その指令がコンパイラによってサポートされていなければ、#pragma 指令は無視され、コンパイルが続けられて正常に終了します。

＜ CC-RX から他の C コンパイラ＞

- CC-RX は、拡張機能としてキーワードの追加を行っているため、他の C コンパイラへ移植するためには、キーワードを削除するか、#ifdef で切り分けなければなりません。

例 1.　キーワードを無効にする

```
#ifndef __RX
#define __evenaccess        /*__evenaccess を指定した変数を通常の変数にします */
#endif
```

例 2.　他のキーワードに変更する

```
#ifdef __RX
_Pragma("inline func")
#else
inline
#endif
void func() { }
```

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.11.28	―	初版発行
1.01	2015.9.14	13, 14	ライセンスの説明を追加しました。
		21, 他	以下のコンパイル・オプションを追加しました。 -misra2012 -stack_protector/-stack_protector_all
		21, 他	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
		149	以下の記述を削除しました。 - utf8 オプションは、lang=c99 オプション指定時のみ有効です。
		152	以下の記述を削除しました。 - outcode=utf8 の指定は、lang=c99 オプション指定時のみ有効です。
		154, 他	アセンブル・オプション -utf8 を追加しました。
		217	最適化リンケージエディタ（rlink）・オプション -crc の機能を拡張しました。
		322, 334	#pragma stack_protector、#pragma no_stack_protector を追加しました。
		804, 他	不要なメッセージを削除しました。
		810 他	以下のメッセージ番号を追加しました。 E0511178 M0523086 W0511179
1.02	2016.07.01	45	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
		219	［詳細説明］を変更しました。
		271, 他	ライブラリジェネレータ・オプション -secure_malloc を追加しました。
		340, 他	組み込み関数に「__」で始まる名称を追加しました。
		344, 他	以下の組み込み関数を追加しました。 __bclr __bset __bnot

Rev.	発行日	改訂内容	
		ページ	ポイント
1.02	2016.07.01	344, 他	組み込み関数の説明を変更しました。
		627-629	以下のライブラリ関数の説明を変更しました。 calloc, free, malloc, realloc
		804, 他	不要なメッセージを削除しました。
		872, 888	以下のメッセージ番号を追加しました。 F0523088 W0520171
1.03	2016.12.01	14	「ライセンスについて」の説明を変更しました。
		14, 15	「standard 版と professional 版について」を追加しました。
		15	「無償評価版について」を追加しました。
		45	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 8.14 9.1 9.4 9.5 12.2 17.6 18.7 21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 21.10 21.11 21.12
		63	［詳細説明］を変更しました。
		73	［備考］を変更しました。
		74	［備考］を変更しました。
		213	［詳細説明］を変更しました。
		229	［詳細説明］を変更しました。
		279	［詳細説明］を変更しました。
		304-311	処理系依存の次の項目を変更しました。 4.1.3 表 4.1, 表 4.3, 表 4.4, 表 4.5, 表 4.7, 表 4.9, 表 4.10, 表 4.11, 表 4.12, 表 4.13, 表 4.14, 表 4.15, 表 4.16
		341	説明を変更しました。
		554, 556, 570	表を変更しました。
		577	［ 戻り値］を変更しました。
		877	F0523073 を変更しました。
		881	F0563020 を追加しました。
		911, 912	「C 標準ライブラリ関数のエラーメッセージ」を追加しました。
		921, 922	「V2.06 以降【V2.05 以前からの変更点】」を追加しました。
1.04	2017.06.01	45	次の MISRA-C:2012 ルールを追加しました。 12.5 13.1 13.2 13.5 17.5 17.8 21.13 21.15 21.16
		23, 48, 66	-avoid_cross_boundary_prefetch オプションを追加しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.04	2017.06.01	57	［詳細説明］を変更しました。
		58	［詳細説明］を変更しました。
		196, 205, 212	-end_record オプションを追加しました。
		207	指定可能なオプションに -show=relocation_attribute と end_record を追加しました。
		232, 233	［指定形式］、［詳細説明］、［備考］を変更しました。
		292	表 3.2 を変更しました。
		293	「リンケージマップ情報」の説明を変更しました。
		294	(7) の説明を変更しました。
		297	(4) の説明を変更しました。
		316	「スカラ型 (C 言語)、基本型 (C++ 言語)」の説明を変更しました。
		323	「Big Endian のメモリ割り付け」の説明を変更しました。
		344	「スタック破壊検出コードを生成する関数の指定」の説明を変更しました。
		810 884, 893, 909	次のメッセージを追加しました。 C0511200、C0519996、C0519997、C0554098、C0564001、F0563103、W0511180、W0511185、W0561015、W0561016
		872, 893, 908, 909	次のメッセージを変更しました。 E0562330、W0511179、W0561004、W0561017
		921-923	「最適化リンケージエディタにおいて最適化有効時の E0562330 エラー発生の回避」を追加しました。
1.05	2017.12.01	22, 28, 38	コンパイル・オプション -no_warning を追加しました。
		23, 49, 68	コンパイル・オプション -insert_nop_with_label を追加しました。
		23, 49, 69-71	コンパイル・オプション -control_flow_integrity を追加しました。
		34	「省略時解釈」を追加しました。
		35	［詳細説明］、［備考］を変更しました。
		79	「省略時解釈」を追加しました。
		80	「省略時解釈」を追加しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.05	2017.12.01	202, 211, 227	最適化リンケージエディタ（rlink）・オプション -fix_record_length_and_align を追加しました。
		202, 211, 236	最適化リンケージエディタ（rlink）・オプション -cfi を追加しました。
		202, 211, 237	最適化リンケージエディタ（rlink）・オプション -cfi_add_func を追加しました。
		202, 211, 238	最適化リンケージエディタ（rlink）・オプション -cfi_ignore_module を追加しました。
		213	No.4 の " 指定可能なオプション " に fix_record_length_and_align を追加しました。
		213	注 3 を変更しました。
		222	［詳細説明］、［備考］ を変更しました。
		226	全体的に説明を変更しました。
		242, 243	表 2.17 に "cfi" を追加しました。
		243, 244	［備考］ の表に "cfi" を追加しました。
		297	表 2.19 に No.30 を追加しました。
		302	表 3.2 に No.10 を追加しました。
		308	「CFI 情報」を追加しました。
		339	表 4.22 に次のマクロを追加しました。 __STDC_IEC_559__、__STDC_IEC_559_COMPLEX__、__STDC_ISO_10646__
		427	表 5.36 に No.8 を追加しました。
		573, 574	<float.h> の定義内容を見直しました。
		820, 881, 882, 891, 892, 894, 899, 917, 918	次のメッセージを追加しました。 E0511117、E0562366、E0563602、F0563003、F0563150、 F0563431、F0563600、F0563601、M0560700、W0561014、 W0561184
		880-882, 891, 892, 898, 918, 920	次のメッセージを変更しました。 E0562311、E0562340、E0562405、E0562417、F0563004、 F0563102、M0560005、M0560103、M0560510、W0561130、 W0561325

Rev.	発行日	改訂内容	
		ページ	ポイント
1.05	2017.12.01	879-920	次のメッセージを削除しました。 E0562021、E0562112、E0562113、E0562203、E0562220、 E0562223、E0562224、E0562323、E0562324、E0562402、 E0562406、E0562407、E0562408、E0562500、F0563311、 F0563312、F0563313、M0560001、M0560512、W0561008、 W0561015、W0561180、W0561500、W0561501、W0561502
		934	「注意」を変更しました。
1.06	2018.06.01	18	ISA_RX の説明を変更しました。
		50	［備考］を変更しました。
		52	「省略時解釈」を追加しました。
		53	「省略時解釈」を追加しました。
		92	［詳細説明］、［備考］を変更しました。
		97	［詳細説明］を変更しました。
		101	［詳細説明］を変更しました。
		116	［指定形式］、［詳細説明］、［備考］を変更しました。
		118	［詳細説明］を変更しました。
		144	［詳細説明］、［備考］を変更しました。
		182	［指定形式］、［詳細説明］、［備考］を変更しました。
		202, 211, 235	最適化リンケージエディタ（rlink）・オプション -split_vect を追加しました。
		213	次の指定可能なオプションを追加しました。 padding、vectn、vect、split_vect
		220	［指定形式］、［詳細説明］、［備考］を変更しました。
		230	［備考］を変更しました。
		233	［詳細説明］を変更しました。
		239	［指定形式］、［詳細説明］、［例］を変更しました。
		316	「関数定義－仮引数の記憶域」の説明を変更しました。
		341, 342	表 4.22 に次のマクロを追加しました。 __RXV3、__RX_ISA_VERSION__
		348	「割り込み関数作成記述」の説明を変更しました。
		382	set_extb、get_extb の［備考］を変更しました。
		406	.RVECTOR の説明を変更しました。
		429	表 5.36 に次のマクロを追加しました。 __RXV3、__RX_ISA_VERSION__
		431	「インストラクション」の説明を変更しました。
		432, 433	「レジスタ構成」の説明を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.06	2018.06.01	465-584	次の命令を追加しました。 BFMOV、BFMOVZ、EMACA、EMSBA、EMULA、FSQRT、FTOU、MACLH、MOVCO、MOVLI、MSBHI、MSBLH、MSBLO、MULLH、MVFACGU、MVFACLO、MVTACGU、RACL、RDACL、RDACW、RSTR、SAVE、UTOF
		488-588	次の命令を変更しました。 FADD、FMUL、FSUB、MACHI、MACLO、MULHI、MULLO、MVFACHI、MVFACMI、MVFC、MVTACHI、MVTACLO、MVTC、POPC、PUSHC、RACW、RTE、RTFI、STNZ、STZ、XOR
		590	表 6.1 を変更しました。
		599, 600	次のリエントラントを変更しました。 fabs、fabsf、fabsl
		659, 660	表 7.10 を変更しました。
		660-662	表 7.11 を変更しました。
		806	「概要」の説明を変更しました。
		806	注 1 を変更しました。
		810	「初期設定ルーチンの記述例」の説明を変更しました。
		810	「低水準インタフェースルーチン」の説明を変更しました。
		827	「リエントラントライブラリ用低水準インタフェースルーチン例」の説明を変更しました。
		831	注 1 を変更しました。
		832	「resetprg.c -- 初期設定ルーチン (リセットベクタ関数)」を変更しました。
		834	【ご参考】を変更しました。
		912, 958, 960	次のメッセージを追加しました。 E0552011、W0561143、W0561331
		862, 912, 922, 928	次のメッセージを変更しました。 E0511154、E0552010、E0562410、F0523073
		879, 928	次のメッセージを削除しました。 E0520412、F0523054、F0523055、F0523056
		975	「コンパイラパッケージのバージョンについて」を追加しました。
1.07	2018.12.01	23, 29, 42	コンパイル・オプション -truncated_address_initializer を追加しました。
		23, 29, 51	コンパイル・オプション -misra_intermodule を追加しました。
		24 65	コンパイル・オプション -nouse_div_inst の説明を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.07	2018.12.01	26, 108	コンパイル・オプション -fpu の説明を変更しました。
		26, 109	コンパイル・オプション -nofpu の説明を変更しました。
		26, 79, 110	コンパイル・オプション -dpfpu を追加しました。
		26, 79, 111	コンパイル・オプション -nodpfpu を追加しました。
		58	コンパイル・オプション -stuff の［詳細説明］を変更しました。
		95	コンパイル・オプション -library の［詳細説明］を変更しました。
		126	コンパイル・オプション -dbl_size の「省略時解釈」を変更しました。
		141	コンパイル・オプション -base の［詳細説明］を変更しました。
		145	コンパイル・オプション -pid の「省略時解釈」、［詳細説明］を変更しました。
		166, 171	アセンブル・オプション -chkpm の説明を変更しました。
		166, 172	アセンブル・オプション -chkfpu の説明を変更しました。
		166, 173	アセンブル・オプション -chkdsp の説明を変更しました。
		166, 168, 174	アセンブル・オプション -chkdpfpu を追加しました。
		166, 180	アセンブル・オプション -fpu の説明を変更しました。
		166, 181	アセンブル・オプション -nofpu の説明を変更しました。
		166, 175, 182	アセンブル・オプション -dpfpu を追加しました。
		166, 175, 183	アセンブル・オプション -nodpfpu を追加しました。
		166, 175, 184	アセンブル・オプション -bank を追加しました。
		166, 175, 185	アセンブル・オプション -nobank を追加しました。
		193	アセンブル・オプション -cpu の［備考］を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.07	2018.12.01	213, 223, 273, 280	最適化リンケージエディタ・オプション -lib_rename を追加しました。
		309	表 2.19 に次のオプションを追加しました。 create_unfilled_area、stack_protector、stack_protector_all、misra2004、misra2012、misra_intermodule
		332	表 4.7 を変更しました。
		352	表 4.22 に次のマクロを追加しました。 __DPFPU
		358, 359, 361	「割り込み関数作成記述」の説明を変更しました。
		372, 373, 395-397	次の組み込み関数を追加しました。 __set_dpsw、__get_dpsw、__set_decnt、__get_decnt、__set_depc、__get_depc
		399	「アセンブラ言語仕様」の説明を変更しました。
		402	「一般命令アドレッシング」の説明を変更しました。
		405	「特定命令アドレッシング」の説明を変更しました。
		442	表 5.36 に次のマクロを追加しました。 __DPFPU
		444	「インストラクション」を削除しました。
		444, 445	表 6.1 に次のセクションを追加しました。 C_8、D_8、B_8
		665	「各種制御レジスタの初期化」の説明を変更しました。
		668	「初期設定ルーチンの記述例」を変更しました。
		690, 691	「resetprg.c -- 初期設定ルーチン (リセットベクタ関数)」を変更しました。
		702	「アプリケーションのコード生成 (pic,pid オプション)」の説明を変更しました。
		724, 800, 814	次のメッセージを追加しました。 E0520069、W0520070、W0551017
		769, 800	次のメッセージを変更しました。 E0523065、W0520069
1.08	2019.04.01	26, 79, 112, 310	コンパイル・オプション -tfu を追加しました。
		353	表 4.22 に次のマクロを追加しました。 __TFU、__TFU_MATHLIB
		363	例 7 を追加しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.08	2019.04.01	374, 402	組み込み関数 __set_depc を削除しました。
		374, 403, 404	次の組み込み関数を追加しました。 __init_tfu、__sincosf、__atan2hypotf
		400-402	__set_dpsw、__get_dpsw、__set_decnt、__get_decnt、__get_depc の [備考] を変更しました。
		462, 463	次の「リエントラント」を変更しました。 atan2、cos、sin、fabs、atan2f、cosf、sinf、fabsl
		464	注 2 を追加しました。
		464	次の「リエントラント」を変更しました。 atan2l、cosl、sinl、sqrtl、fabsl
		468	注 1、注 2 を追加しました。
		483, 497	ライブラリ関数 hypot、hypotf、hypotl の説明を変更しました。
		549, 550	bsearch、qsort の [指定形式] を変更しました。
		674	「三角関数演算器の初期化」を追加しました。
		677	「初期設定ルーチンの記述例」を変更しました。
		778, 790, 795	次のメッセージを追加しました。 E0523129、E0562600、F0523129

Rev.	発行日	改訂内容	
		ページ	ポイント
1.09	2019.11.01	12	「著作権について」の説明を変更しました。
		48	コンパイル・オプション -misra2012 の［備考］を変更、［使用例］を追加しました。
		48	次の MISRA-C:2012 ルールを追加しました。 8.13、14.2、14.3
		24, 52, 57	コンパイル・オプション -g_line を追加しました。
		112	コンパイル・オプション -tfu の［詳細説明］を変更しました。
		213, 216, 223, 226, 281	最適化リンケージエディタ（rlink）・オプション -ALLOW_DUPLICATE_MODULE_NAME を追加しました。
		312	表 2.19 を変更しました。
		332	「処理系依存」を削除し、「C90 の処理系定義」を追加しました。
		336	「C99 の処理系定義」を追加しました。
		366, 367	「セクション切り替え記述」の説明を変更しました。
		505, 506	関数の説明を変更しました。
		562	表の「定義名」に HUGE_VALF/HUGE_VAL/HUGE_VALL を追加しました。
		695	cacosf / cacos / cacosl の [戻り値] と [備考] を変更しました。
		696	casinf / casin / casinl の [戻り値] と [備考] を変更しました。
		697	catanf / catan / catanl の [戻り値] を変更しました。
		701	cacoshf / cacosh / cacoshl の [戻り値] と [備考] を変更しました。
		732	imaxdiv の説明を全体的に変更しました。

Rev.	発行日	改訂内容	
		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。
		11, 217, 281	誤記を訂正しました。（スタック情報ファイル）
		18	表 2.1 にツール使用情報ファイルを追加しました。
		26, 80, 84, 117	コンパイル・オプション -branch_chaining を追加しました。
		26, 80, 83, 84, 118	コンパイル・オプション -nobranch_chaining を追加しました。
		32	コンパイル・オプション -preinclude の［詳細説明］を変更しました。
		130	コンパイル・オプション -dbl_size の［詳細説明］を変更しました。
		217, 279, 295	最適化リンケージエディタ（rlink）・オプション -verbose を追加しました。
		914	表 9.1 に次のレジスタを追加しました。 DCMR、DPSW/DECNT/DEPC
		975	次のメッセージを追加しました。 E0552020
		1038	「注意事項」を追加しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.11	2021.11.01	10	「概説」の説明を変更しました。
		18	「環境変数」の説明を変更しました。
		25, 80, 92	コンパイル・オプション -type_size_access_to_volatile を追加しました。
		83	コンパイル・オプション -speed の［備考］を変更しました。
		84	コンパイル・オプション -size の［備考］を変更しました。
		86	コンパイル・オプション -inline の［備考］を変更しました。
		87	コンパイル・オプション -noinline の［備考］を変更しました。
		102	コンパイル・オプション -map の［詳細説明］を変更しました。
		139	コンパイル・オプション -pack の説明を全体的に変更しました。
		140	コンパイル・オプション -unpack の説明を全体的に変更しました。
		216, 220, 227	オプション名の表記を小文字に変更しました。 -allow_duplicate_module_name
		264	最適化リンケージエディタ（rlink）・オプション -optimize の［備考］を変更しました。
		344	「C99 の処理系定義」(47) の説明を変更しました。
		368	「#pragma 指令」に説明を追加しました。
		377, 378, 382	「拡張仕様の使用方法」の説明を変更しました。
		384	「キーワードの使用方法」の説明を変更しました。
		466	「セクション名一覧」の説明を変更しました。
		671, 673, 675, 737	次の関数の説明を変更しました。 div、ldiv、lldiv、imaxdiv
		926, 998, 1005, 1021	次のメッセージを変更しました。 E0511178、F0563430、W0511180、W0511185、W0561016、W0561017
		966	次のメッセージを削除しました。 E0521212
		1000, 1001	次のメッセージを追加しました。 M0520177、M0520550、M0520826
		1044	「関数のインライン展開を行う」の説明を変更しました。
		1045	「C 言語で CPU 命令を使用する」の説明を変更しました。
		1065	「C ソースの修正」の説明を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.12	2022.12.01	26, 80, 116	コンパイル・オプション -tfu_version を追加しました
		26, 80, 117	コンパイル・オプション -nosave_tfu を追加しました。
		67, 194, 261, 313, 371, 474 他	対象リビジョンを追記しました。
		114, 115	コンパイル・オプション -tfu の［詳細説明］、［備考］を変更しました。
		157	コンパイル・オプション -save_acc の［備考］を変更しました。 MACW 命令→ MACLO 命令
		228	最適化リンケージエディタ（rlink）・オプション -entry の［備考］を変更しました。
		248-250	最適化リンケージエディタ（rlink）・オプション -crc の［詳細説明］、［例］を変更しました。
		321	表 2.19 の " 無効とされるオプション " に次のオプションを追加しました。 tfu_version、nosave_tfu
		340	「未定義の動作」(23) の説明を変更しました。
		343	「C90 の処理系定義」(25) の説明を変更しました。
		357	表 4.1 の " データメンバへのポインタ " の符号の有無を変更しました。
		371	表 4.6 の「プリデファインドマクロ」欄を変更しました。
		378, 379	「割り込み関数作成記述」の説明に以下の割り込み仕様を追加しました。 tfu、no_tfu
		394, 426-429	次の組み込み関数を追加しました。 __sincosfx、__sinfx、__cosfx、__atan2hypotfx、__atan2fx、__hypotfx
		423	組み込み関数 __init_tfu の [機能]、[例]、[備考] を変更しました。
		424	組み込み関数 __sincosf の [例]、[備考] を変更しました。
		425	組み込み関数 __atan2hypotf の [例]、[備考] を変更しました。
		489, 493	表 7.4 および表 7.5 の注記を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.12	2022.12.01	752	<stdint.h> の表の「説明」欄を変更しました。
		884	「初期設定ルーチンの記述例」のコーディング例を変更しました。
		984, 994, 1001, 1006	次のメッセージを追加しました。 E0523049、E0562220、F0520571、F0563115
		1031	次のメッセージを変更しました。 W0561017
1.13	2023.12.01	10	「概要」の説明を変更しました。
		14	表 1.2 と注記を変更しました。
		16	「概要」の「プリプロセッサ」の説明を変更しました。
		18	表 2.2 の「設定省略時の解釈」欄を変更しました。
		33	コンパイル・オプション -define の［詳細説明］を変更しました。
		41	コンパイル・オプション -comment の［詳細説明］を変更しました。
		51	コンパイル・オプション -misra_intermodule の［備考］を変更しました。
		62	コンパイル・オプション -instalign4 の［詳細説明］を変更しました。
		63	コンパイル・オプション -instalign8 の［詳細説明］を変更しました。
		77, 79	参照先のタイトルに章節番号を追加しました。
		98	コンパイル・オプション -scope の「省略時解釈」を変更しました。
		99	コンパイル・オプション -noscope の「省略時解釈」を変更しました。
		102	コンパイル・オプション -map の「省略時解釈」を変更しました。
		105	コンパイル・オプション -nomap の「省略時解釈」を変更しました。
		221, 267, 275	最適化リンケージエディタ（rlink）・オプション -ALLOW_OPTIMIZE_ENTRY_BLOCK を追加しました。
		239	最適化リンケージエディタ（rlink）・オプション -rom の［詳細説明］を変更しました。
		249	最適化リンケージエディタ（rlink）・オプション -crc の［詳細説明］を変更しました。
		253	最適化リンケージエディタ（rlink）・オプション -padding の［詳細説明］を変更しました。
		300	最適化リンケージエディタ（rlink）・オプション -total_size の［備考］を変更しました。
		309	「ライブラリジェネレータ・オプション」に説明を追加しました。
		321, 322	表 2.19 を変更しました。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.13	2023.12.01	325	「オブジェクト情報」内の表を変更しました。
		344	「C90 の処理系定義」(33) の説明を変更しました。
		358, 363, 622	参照先の表番号を修正しました。
		371, 372	表 4.6 と注記を変更しました。
		386, 387	「拡張仕様の使用方法」の「構造体 / クラスメンバのアライメント指定」および「変数の絶対アドレス割り付け指定」の説明を変更しました。
		395, 425-427, 429	引数名を下記のように変更しました。 sin → s cos → c atan2 → a hypot → h
		475	表 5.36 と注記を変更しました。
		488	表 7.4 を変更しました。
		491-495	表 7.5 を変更しました。
		520	「<math.h>」内の表から FP_FAST_FMA/FP_FAST_FMAF/FP_FAST_FMAFL の行を削除しました。
		525	acos / acosf / acosl 関数の［備考］を変更しました。
		526	asin / asinf / asinl 関数の［備考］を変更しました。
		527	atan / atanf / atanl 関数の［備考］を変更しました。
		528	atan2 / atan2f / atan2l 関数の［備考］を変更しました。
		531	tan / tanf / tanl 関数に［備考］を追加しました。
		882	「スタートアップ」の「概要」の説明を変更しました。
		935, 937, 947, 949, 950, 964, 967, 968, 971, 973, 979, 980, 988, 991, 1001-1003, 1008, 1019, 1020, 1022, 1032	次のメッセージを変更しました。 C0519996、E0511104、E0520170、E0520172、E0520257、E0520276、E0520676、E0520767、E0520791、E0520870、E0520940、E0521255、E0521282、E0532002、E0552063、E0592001、E0592002、E0592003、E0592004、E0592005、E0592006、E0592007、E0592008、E0592010、E0592013、E0592015、E0592016、E0592018、E0592019、E0592020、E0592101、E0592102、E0592201、E0593002、E0593003、E0593004、E0594000、E0594001、E0594002、F0512003、F0520003、F0553200、W0520001、W0520027、W0520181、W0521607

Rev.	発行日	改訂内容	
		ページ	ポイント
1.13	2023.12.01	935, 936, 996, 1012, 1013, 1039	次のメッセージを追加しました。 C0520000、C0529000、C0590001、E0562114、F0593021、F0593300、F0593302、F0593303、F0593305、F0593320、F0593321、F0593322、F0593324、F0593325、F0593326、F0593327、W0591300、W0591301
		939	次のメッセージを削除しました。 E0511165
		1052, 1053	「V2.06 以降【V2.05 以前からの変更点】」の説明を変更しました。
		1056	「値を変更しない初期化変数は const 宣言をする」の説明を変更しました。

CC-RX ユーザーズマニュアル

発行年月日　2014年 11月 28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-RX

RENESAS

ルネサス エレクトロニクス株式会社

R20UT3248JJ0113